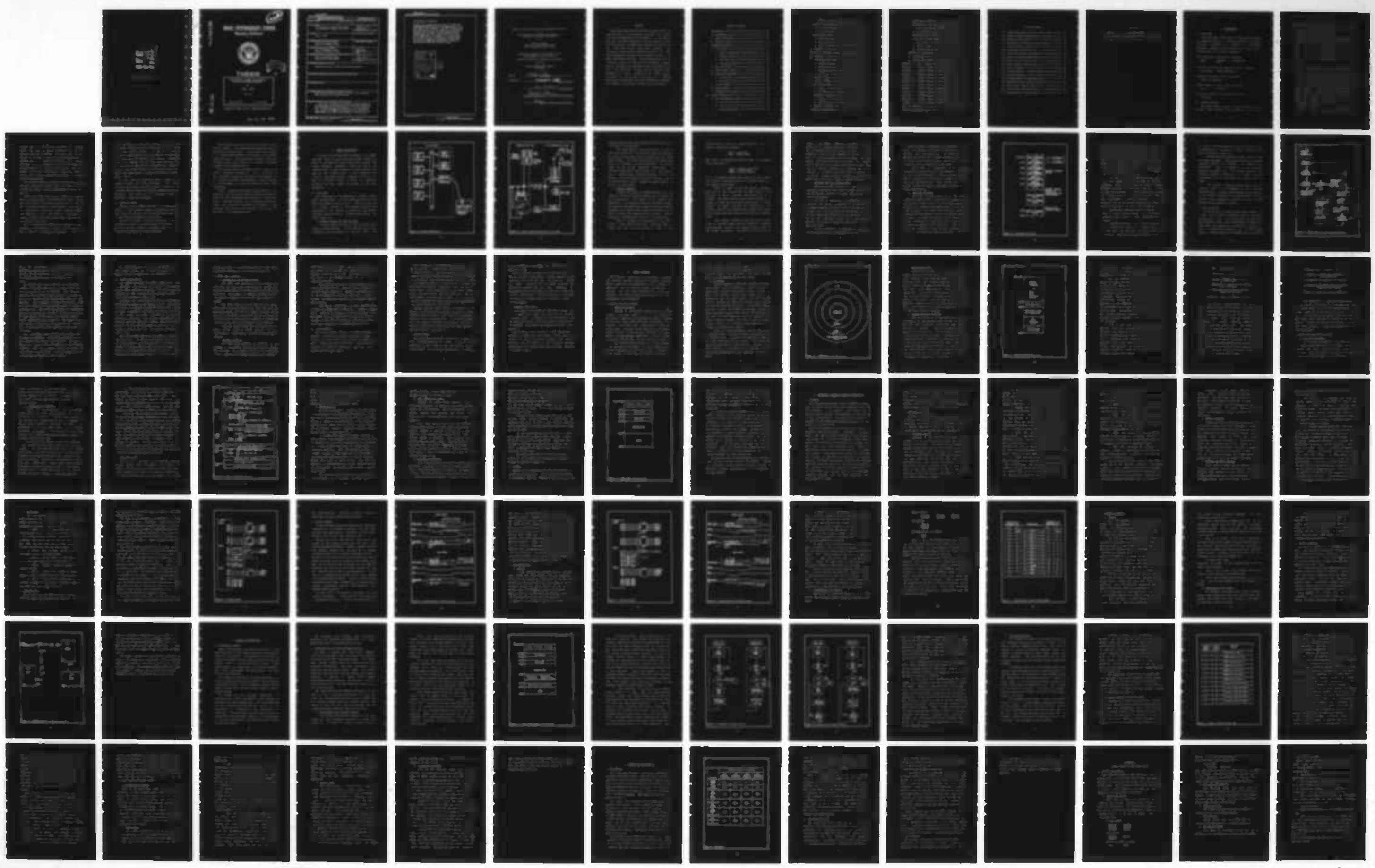
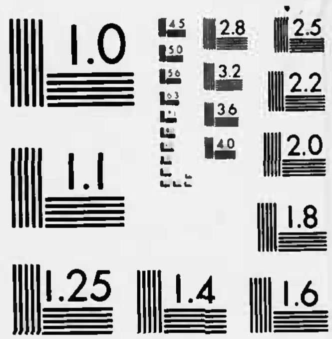
AD-A132 165 LOGIC DESIGN OF A SHARED DISK SYSTEM IN A MULTI-MICRO COMPUTER ENVIRONMENT(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA M L PERRY JUN 83

1/3

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A132165

NAVAL POSTGRADUATE SCHOOL

Monterey, California

DTIC ELECTE SEP 7 1983 S D

THESIS

D

LOGIC DESIGN OF A SHARED DISK SYSTEM IN A MULTI-MICRO COMPUTER ENVIRONMENT

by

Mark L. Perry

June 1983

Thesis Advisor: M. L. Cotton

Approved for public release; distribution unlimited

DTIC FILE COPY

83 09 06 060

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO. AD-A132 165

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
Logic Design of a Shared Disk System in a Multi-Micro Computer Environment

5. TYPE OF REPORT & PERIOD COVERED
Master's Thesis
June 1983

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Mark L. Perry

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Naval Postgraduate School
Monterey, California 93940

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Naval Postgraduate School
Monterey, California 93940

12. REPORT DATE
June 1983

13. NUMBER OF PAGES
200

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)
Naval Postgraduate School
Monterey, California 93940

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
multi-user, Micropolis, CP/M-86, hard disk, disk interface, AEGIS, parallel I/O, shared resources.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis describes the detailed interface design and implementation of the Micropolis 1220 rigid disk storage unit into the AEGIS multiuser environment. At the onset of this work, the AEGIS development system consisted of an MBB-80 bubble memory, the REMEX Data Warehouse disk system, and four INTEL iSBC 86/12A single board computers. The Micropolis

DD FORM 1473, 1 JAN 73 — EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(continuation of abrtact)

interface was accomplished utilizing the INTEL 8255 programmable parallel I/O port resident on one of the AEGIS iSBC 86/12A computers. The iSBC 86/12A used for the interface can still be operated as an independent computer with all Micropolis disk operations being transparent to the user. The Micropolis disk unit adds an additional 35.6 megabytes of online storage to the AEGIS system. Utilization of the Micropolis disk system as a software development storage media will free the REMEX Data Warehouse for storage of "radar data" to emulate the SPY-1A radar.

Approved for public release; distribution unlimited

Logic Design of a Shared Disk System in a
Multi-Micro Computer Environment

by

Mark L. Perry
Captain, United States Army
B.S., Purdue University, 1976

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 1983

Author: Mark L. Perry

APPROVED BY: Mitchell L. Cotton
Thesis Advisor

Uno R. Kodres
Second Reader

Robert D. Strum
Chairman, Department of Electrical Engineering

J.M. Dyer
Dean of Science and Engineering

ABSTRACT

This thesis describes the detailed interface design and implementation of the Micropolis 1220 rigid disk storage unit into the AEGIS multiuser environment. At the onset of this work, the AEGIS development system consisted of an MBB-80 bubble memory, the REMEX Data Warehouse disk system, and four INTEL iSBC 86/12A single board computers. The Micropolis interface was accomplished utilizing the INTEL 8255 programmable parallel I/O port resident on one of the AEGIS iSBC 86/12A computers. The iSBC 86/12A used for the interface can still be operated as an independent computer with all Micropolis disk operations being transparent to the user. The Micropolis disk unit adds an additional 35.6 megabytes of online storage to the AEGIS system. Utilization of the Micropolis disk system as a software development storage media will free the REMEX Data Warehouse for storage of "radar data" to emulate the SPY-1A radar.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

A. DISCLAIMER

Many terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each individual occurrence of a trademark, all registered trademarks appearing in this thesis will be listed below, following the firm holding the trademark.

Intel Corporation, Santa Clara, California:

INTEL	MULTIBUS	INTELLEC MDS
iCS	iSBC	

Pacific Cyber/Metrixs Incorporated, Dublin, California:

Bubbl-Board MBB-80 Bubbl-Board

EX-CELL-O Corporation, Irvine, California:

REMEX Data Warehouse

Digital Research, Pacific Grove, California:

CP/M CP/M-86

Micropolis Corporation, Chatsworth, California:

Micropolis

B. GENERAL DISCUSSION

The AEGIS weapons system simulation project is an ongoing study currently being conducted at the Naval

Postgraduate School. The primary objective of this study is to investigate the feasibility of replacing the present, four-processor AN/UYK-7 mainframe computer with a multiple microcomputer based architecture.

The primary mission of the multiprocessor system is to provide computer control for the SPY-1A radar system. This system collects large amounts of data concerning target detection and acquisition which must be processed in real-time. A microcomputer based system can provide the same signal processing in real-time only if more than one processor is used and the computations are performed concurrently.

Thus, the first logical step of the AEGIS study was to identify a viable microcomputer and design an efficient operating system capable of managing concurrent processes. A detailed design of such an operating system was presented by Wasson in 1980 [Ref. 1]. This design was based on the INTEL iSBC 86/12A microcomputer. This is a single board computer based on the INTEL 8086 16-bit microprocessor. The operating system, MCORTEX, was implemented using Wasson's design and refined many times. Klinefelter demonstrated the first truly efficient implementation of MCORTEX with four iSBC 86/12A's in June of 1982 [Ref. 2].

Because MCORTEX was a very specialized manager of concurrent processes, it was not an operating system well suited for program module development. Thus, the next

logical step of the project was to identify an operating system that could be easily integrated into the same hardware utilized by MCORTEX. This would allow the same system to be used for both signal processing emulation by MCORTEX and as a software development tool. CP/M-86, developed by Digital Research for use with the INTEL 8086 microprocessor, was chosen for this purpose. This choice offered the maximum in flexibility in that this operating system could be user configured for different or changing hardware environments.

Mike Candalor began the integration process by modifying the Basic Input/Output System (BIOS) of CP/M-86 for use on an INTEL MDS system. This was demonstrated in June of 1981. [Ref. 3]

Hicklin and Neufeld continued the integration process by adding a bubble memory to the MULTIBUS and again altering the BIOS to reflect the current hardware [Ref. 4]. Due to the non-volatile nature of a bubble memory, it was used in this application to store the CP/M-86 operating system. This permitted a fast, easy method of downloading the operating system into random access memory (RAM) when power was applied to the system.

Since the Klinefelter demonstration employed simulated processes, it was necessary to develop a method by which the SPY-1A radar could be emulated in real-time. A hard disk

drive, interfaced through direct memory access (DMA), was determined suitable for this purpose due to its high speed and large storage capacity. It was also considered desirable to make maximum use of the available hardware when the system was being operated in the software development mode. This required that each single board computer have the capability of supporting an independent user. These two concepts were brought together and demonstrated with a four-board, multi-user system by Almquist and Stevens in December 1982. [Ref. 5]

At this point the system still lacked a capability of storing software for future refinement. This thesis completes the program development system by presenting the hardware interface design and software implementation of the Micropolis disk drive into the multi-user system as developed by Almquist and Stevens.

C. FORMAT OF THESIS

Chapter I gives a general overview of the AEGIS research effort. It also provides a general developmental history of the project and explains why the research work accomplished by this thesis was essential to the project.

Chapter II addresses the system architecture. Detailed discussion is given of all major hardware components as this was the existing hardware environment into which the Micropolis disk drive had to be interfaced.

Chapter III describes the details of the AEGIS multiuser system software. The standard CP/M-86 operating system is discussed in moderate detail. Also covered in this chapter is a powerful modification to CP/M-86 that was developed during prior work. This modification provides a simple and efficient method for altering the hardware environment supported by the operating system.

The hardware interface developed for the Micropolis disk system is presented in Chapter IV. First, the details of the requirements imposed on the design of the interface by the Micropolis controller are presented. This is followed by the development of a functional interface to meet those requirements.

Chapter V presents the software implementation of the Micropolis into the CP/M-86 operating system and Chapter VI summarizes the development work accomplished during this thesis. Included in Chapter VI is a comparison of the disk access times required by the REMEX Data Warehouse (a DMA interfaced hard disk) and the Micropolis disk system (a programmed I/O interfaced hard disk).

II. SYSTEM ARCHITECTURE

As stated in the introduction, the design of the MCORTEX operating system and the software development system was based on the INTEL iSBC 86/12A single board computer and various peripheral components. Figure 2.1 depicts the interconnection of these components as they existed at the onset of this research effort. In the paragraphs that follow, a description of each component, as well as its role in the overall system, is given. An exhaustive description of each device can be found in the cited references.

A. INTEL 8086

The INTEL 8086 is a high performance, general purpose 16-bit microprocessor. It is the foundation upon which the AEGIS developmental system is built. Refer to Figure 2.2 for a general overview of its internal structure and organization. This section is intended to give general knowledge about this device. A detailed description can found in [Ref. 6].

1. General Purpose and Flags Registers

As can be seen in Figure 2.2, there are eight 16-bit general purpose registers. Four of these are byte or word addressable and are referred to as "the data group". The

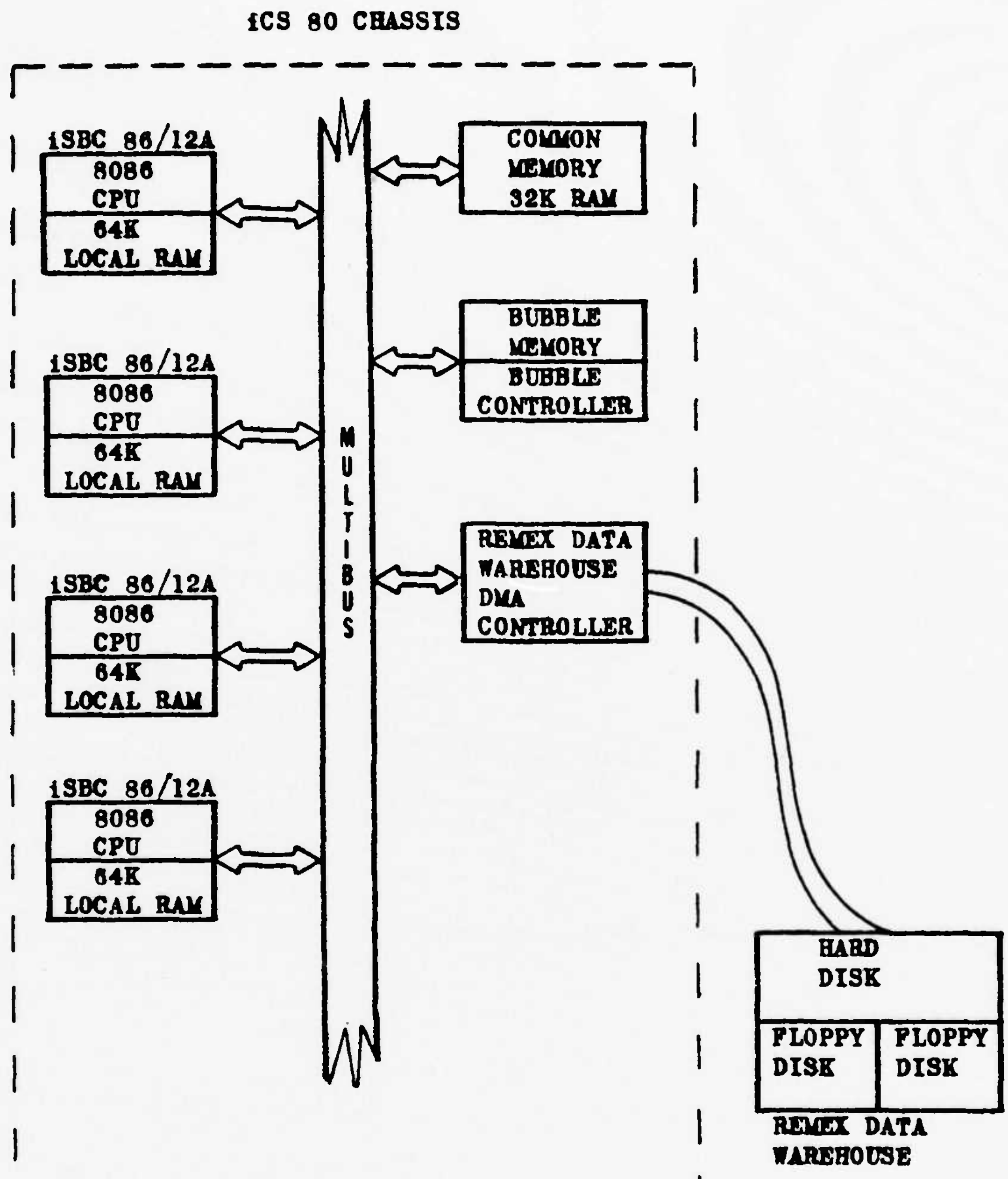


Figure 2.1 System Configuration

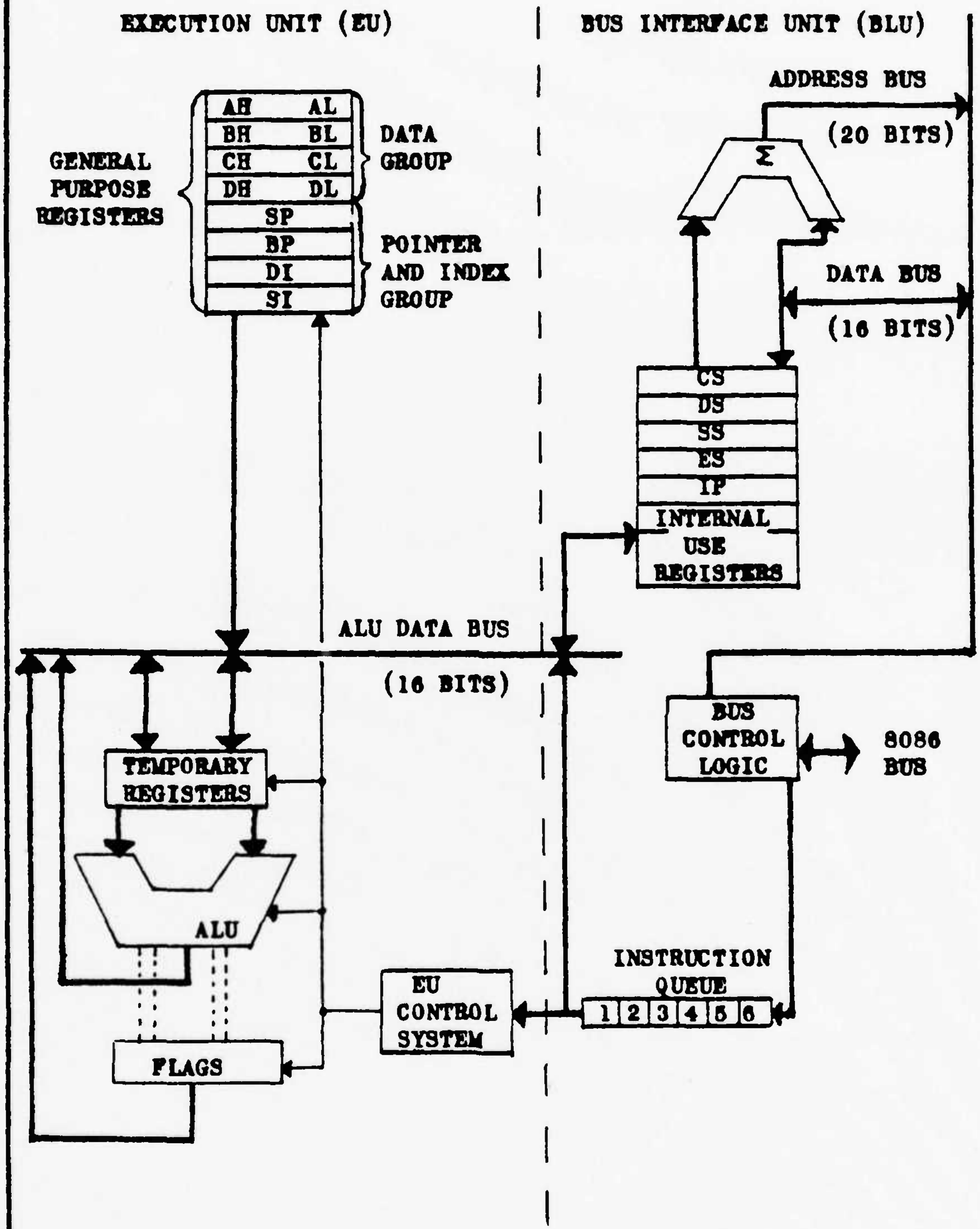


Figure 2.2 General Internal Organization of the 8086

remaining four are only word addressable and are referred as "the pointer and index group".

The flags register is 16 bits wide and consists of nine usable status bits. The remaining seven are undefined. The nine bits are divided into six status flags and three control flags. The status flags are set by the 8086 as the result of arithmetic or logical operations. The control flags are set through programmed instructions. Of particular importance is the IF control flag. This flag is used to enable/disable maskable interrupts and must be properly set for the system to function correctly. The IF or interrupt-enable flag is discussed in greater detail in Section 4 of this chapter.

2. <u>Segment Registers</u>

Although the 8086 has segment registers and the technical literature discusses segmentation as related to the microprocessor, this should not be confused with segmentation as is generally defined for operating systems. The operating system definition supports the ideas of memory management and segment access checks but the 8086 has no special hardware that supplies these functions. However, addressing is segment-like in that it is two-dimensional.

Physical addresses are generated from two 16-bit values: a base and an offset value. The base value is shifted left four bits and the offset is added to this

shifted version to arrive at a physical address. As an example, consider the following:

```
E000 -- BASE VALUE
2AAA -- OFFSET VALUE
```

When these two hexidecimal values are added as described above, the result is:

```
E0000 -- SHIFTED BASE VALUE
 2AAA -- OFFSET VALUE
E2AAA -- PHYSICAL ADDRESS
```

It is the segment registers that supply the base value. This method of addressing results in a 20-bit address or a one megabyte address space.

Shown in Figure 2.2 are four segment registers. Each are 16 bits wide and give the 8086 access to 64 kilobytes of memory. Assuming they are each set to a different 64K base, this will give the CPU access to a maximum of 256K bytes of memory at any instant of time. Because the segment registers are accessible to the software, they can be programmatically altered to any value. Thus allowing addressing throughout the entire one megabyte range.

Which segment register is used and how the offset value is obtained depends upon the instruction currently being executed. The CS or code segment register points to the current code segment. All executable instructions are

located in this segment. Therefore, the address of the next instruction is computed using the CS register as the base and the instruction pointer (IP) as the offset. All stack operations utilize the stack segment (SS) register as the base and the stack pointer (SP) as the offset. The data segment (DS) register and the extra segment (ES) register have no explicit offset register associated with them. This value is software controlled by supplying one of the registers in the pointer and index group as a part of the instruction. Program variables are generally placed in the memory space accessible by these two segment registers.

3. <u>Execution and Bus Interface Units</u>

The dividing line in Figure 2.2 is used to indicate two separate processing units within the 8086: the execution unit (EU) and bus interface unit (BIU). Both of these units operate independently.

The BIU is responsible for performing all bus operations for the EU. It generates 20 bit addresses by combining the segment and offset values in its own adder and transfers data to and from the EU on the ALU data bus. The BIU also fetches instructions for the EU and stores them in its six byte instruction queue. This queue makes it possible for the BIU to "prefetch" instructions during any spare bus cycles.

The EU is responsible for executing all instructions and for transfering data and addresses to the BIU. It also

maintains the general purpose and flags registers. Because there is no connection from the EU to the system bus, it is isolated from the outside world. All instructions to be executed are fetched from the BIU's instruction queue. In the event that the queue is empty, the EU simply waits for the BIU to place an instruction in the queue.

This type of architecture allows extensive overlapping of instruction fetch with execution. The result is that the time required to fetch instructions becomes nearly transparent to the EU since it works on instructions that have been prefetched. This, coupled with a 5 MHZ clock, gives the 8086 the high speed necessary for the AEGIS implementation.

4. Interrupt Structure

The 8086 has provisions for up to 256 different interrupts numbered from 0 to 255. When an interrupt occurs, the CPU must transfer control to a new program location that contains the necessary programmed instructions to service that interrupt. Two values are necessary to effect the transfer: the code segment in which the interrupt routine is located and the instruction pointer for the routine. These values are located in a table that begins at absolute zero in memory and extends to 3FF hexidecimal. Refer to Figure 2.3. The information needed for each interrupt routine occupies four consecutive bytes in this table. The CPU is

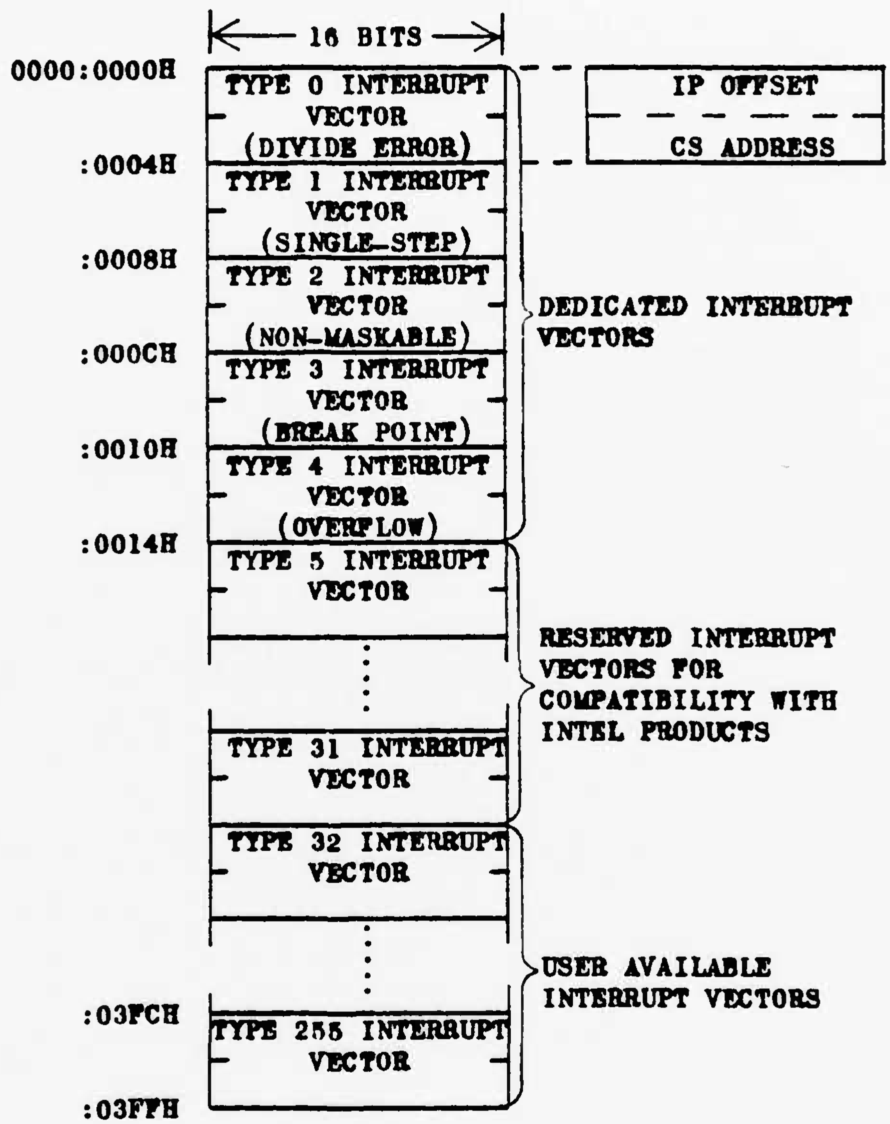


Figure 2.3 Interrupt Vector Table

supplied with a type code when an interrupt occurs. This value is automatically multiplied by four to determine the correct position in the table from which to obtain the CS and IP values. The current CS, IP and flags register values are pushed on the stack and the new CS and IP values are loaded. This completes the transfer of control. Both the values in the interrupt table and the interrupt routines are user supplied and must be placed in memory before the interrupt can be permitted to occur.

How the processor is supplied with the type code cited above depends on the method used to generate the interrupt. These can be software or hardware generated. Hardware interrupts are subdivided into two categories: maskable and non-maskable. Maskable interrupts are enabled or disabled by setting or clearing the IF flag. When the CPU acknowledges a maskable interrupt, it is the responsibility of the hardware requesting the interrupt to place the type code on the bus for use by the CPU.

Non-maskable interrupts cannot be disabled. In the event of a non-maskable interrupt, the CPU automatically assigns it a type code of 2. Thus, a type code need not be supplied.

Software interrupts can be invoked by executing the "INT n" instruction; where "n" is a number from 0 to 255. In this case, the type code is an explicit part of the instruction. They can also occur by creating a fatal error

as a result of program execution, such as a divide by zero or overflow error. The CPU will then use a predefined type code as depicted in Figure 2.3.

The 8086 does not generally control the devices that can cause interrupts. This makes simultaneously occurring interrupts possible and therefore, all interrupts are prioritized. Shown in Figure 2.4 is the interrupt processing sequence used. This figure indicates that software generated interrupts are the highest priority. Non-maskable are the next highest and maskable are the lowest.

The interrupt structure discussed above plays an important role in the development of the Micropolis interface design. A general understanding of this structure is an essential prerequiste to an understanding of both the detailed design presented in Chapter IV and the software implementation presented in Chapter V.

B. THE iSBC 86/12A

The iSBC 86/12A is a complete single board computer. It is used as the central processing node of the AEGIS multiprocessor system. The board includes the 8086 16-bit CPU, 64K bytes of RAM, a serial communications interface, an INTEL 8255 that supplies three programmable parallel I/O ports, an INTEL 8253 programmable timer, an INTEL 8259A priority interrupt controller, MULTIBUS interface control

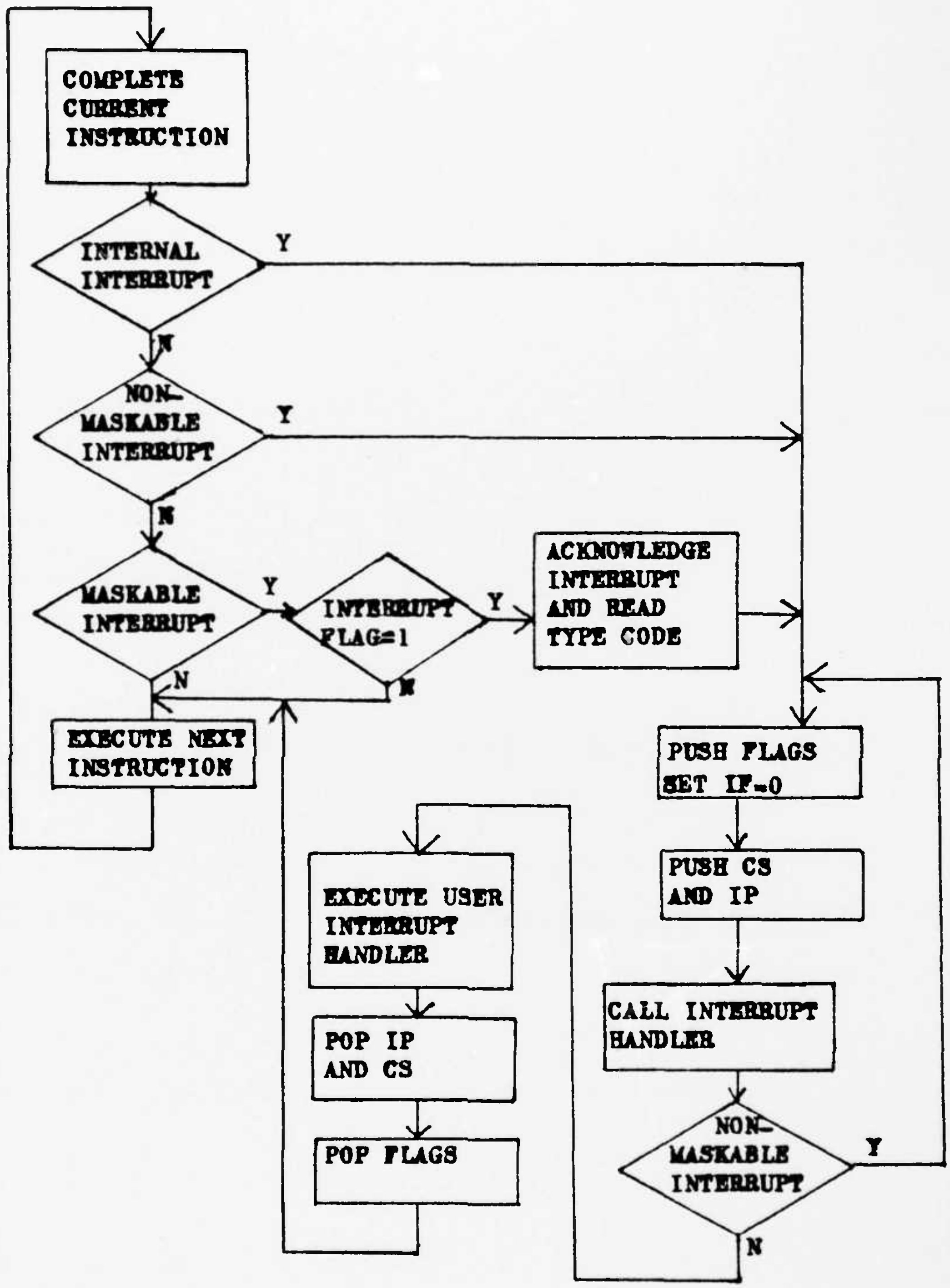


Figure 2.4 Generalized Interrupt Processing Sequence

logic, and bus expansion drivers for interface with other MULTIBUS interface-compatible expansion boards. Provisions are also made for installation of up to 16K bytes of EPROM. The current system only utilizes 8K bytes of EPROM on each board.

The onboard 64K bytes of RAM can be dual-ported in segments of 16K bytes thus making it accessible not only to the local CPU but also to the MULTIBUS. When dual-ported, the RAM can be switch-and-jumper configured to any 128K byte segment of the one megabyte address space relative to the MULTIBUS. Local addresses are always fixed between 0000H and FFFFH regardless of the MULTIBUS address the board is configured for. This system was designed for independent operation by each SBC and therefore, no RAM is dual-ported. To make the RAM inaccessible to the MULTIBUS requires a jumper between E112-E114 on each SBC. The board does not come factory equipped with this jumper and its existance must be verified before proper operation of the system can be insured.

Each iSBC comes factory configured with jumpers between E103-E104 and E105-E106. These jumpers route the bus clock and constant clock signals to the MULTIBUS. As shown in Figure 2.1, several SBC's are connected to the MULTIBUS interface. Only one of these is required to supply the clock signals to the MULTIBUS. All other boards must have the E103-E104 and E105-E106 jumpers removed.

No other special configurations are necessary for the iSBC 86/12A boards. For a complete discussion of user options and factory defaults for this board see [Ref. 7].

C. MBB-80 BUBBLE MEMORY

The MBB-80 Bubbl-Board is a memory storage device that is compatible with all 8-bit and 16-bit microprocessors having INTEL MULTIBUS architecture. The board provides approximately 90K bytes of non-volatile memory as well as all required MULTIBUS interface logic.

Interface with the MBB-80 controller is accomplished through memory mapped I/O and requires sixteen user-defined locations in the MULTIBUS one megabyte address space. These addresses correspond to controller registers that are used to read status, set board configurations and perform read/write operations. The current configuration uses MULTIBUS addresses beginning at 80000H. This requires that switch 8 in S2 on the Bubbl-Board be set to "on" and all others in S2 be set to "off". All switches in S1 must be set to "off".

The bubble memory appears to the system as a simple 90k byte disk drive. All read/write operations to this device are accomplished in the same manner used for any other disk system and require no special user invoked functions. Its primary use in the system as depicted in Figure 2.1 is as a non-volatile storage medium from which to load the operating

system into RAM. For a complete discussion of the MBB-80 Bubbl-Board implemetation to the system see [Ref. 4].

D. REMEX DATA WAREHOUSE

1. Subcomponents and Storage Capacity

The REMEX Data Warehouse is a mass storage unit containing two floppy disk drives (single or double-sided, single or double density) and a Winchester technology fixed disk drive. Additionally, an MC6800 microprocessor coupled with onboard firmware is the mechanism used to service all drives.

The fixed disk is a 14 inch enclosed disk system consisting of two recording surfaces. Each surface has two reccording heads. Each head can access 210 usable tracks and each track contains 39 512-byte sectors. This gives each head access to approximately 4 megabytes of storage and gives the disk a total storage capacity of 16 megabytes.

The two floppy drives are switch-selectable to handle either single or double density. In this implementation, single density, standard IBM FM encoding is employed.

2. MULTIBUS Interface

The REMEX Data Warehouse is interfaced to the MULTIBUS via the MULTIBUS Interface Card assembly supplied with the unit. This assembly contains all the necessary control, buffering and MULTIBUS interface logic required to

interface with the host system. The host communicates with the assembly using programmed I/O. Communications from the assembly to the host is via DMA. The interface acts as a bus master in the DMA mode and as a bus slave in the programmed I/O mode.

The controller requires 4 I/O port addresses for the host system communications. These are used to obtain status and pass command information. Currently, addresses 70, 71, 72 and 73 hexidecimal are used but these can be altered by changing the appropriate switches on the MULTIBUS Interface Card assembly.

The system configuration in this implementation utilizes the REMEX Data Warehouse as a program storage media. However, as alluded to in the introduction, it is envisioned that this hard disk drive will be used for storage of track data in the SPY-1A radar emulation effort. For further information on the REMEX consult [Ref. 8].

E. iCS-80 CHASSIS

The iCS-80 industrial chassis is MULTIBUS-compatible and supports a modular microcomputer development system. It consists of four four-slot iSBC 604/614 Cardcages, four fans, a power supply and control panel. The control panel contains an on/off/lock key switch, reset and interrupt pushbuttons and various LED's.

Any combination of MULTIBUS-compatible plug in boards can be installed. A maximum of four boards can be placed in the iSBC 604 Cardcage. Additional iSBC 614 Cardcages can be added to the chassis through an expansion interface supplied with the system. The laboratory system utilized in support of this thesis consists of a single iSBC 604 Cardcage and three iSBC 614 Cardcages. This gives a total capacity of 16 board slots. These cages provide for both INTEL MULTIBUS master and slave boards. From the front panel, the slots are numbered 1 to 16 from left to right. All odd-numbered slots are configured for master boards and all even-numbered slots are configured for slave boards.

Because more than one bus master can be placed in the chassis, a priority resolution scheme is required to resolve MULTIBUS access contention. This scheme can be operated in either the serial or parallel mode. In the system of Figure 2.1, the chassis is operated in the parallel mode with an external random priority network for bus access resolution. For more information see [Ref. 9].

F. COMMON MEMORY

The common memory depicted in Figure 2.1 is a simple 32K byte, MULTIBUS-compatible RAM board. It can be switch configured to any address in the one megabyte Multibus address space. In its current configuration, it occupies

addresses E0000H through E7FFFH. The board is expandable to 64K bytes of RAM.

Recall from the discussion on the iSBC 86/12A that the RAM of all SBC's in the system is jumper configured to be accessible only to the local CPU. This means that neither the bubble controller nor the REMEX controller can communicate directly with any iSBC. Therefore, all read/write operations with these two devices is accomplished via the common memory. The technique used to coordinate this effort is a software one and is discussed in detail in the next chapter.

G. MICROPOLIS DISK DRIVE

The Micropolis disk system (not depicted in Figure 2.1) is an eight inch fixed disk drive with an integral controller board. It consists of five data surfaces with 580 tracks per surface. Each track contains twenty-four 512 byte sectors. This gives a 35.6 megabyte formatted storage capacity.

The controller board consists of a Z-80 microprocessor, firmware in PROM, and the necessary control logic and buffers to provide a variety of features. The features employed and the details of the Micropolis interface accomplished as a result of this thesis work are presented in Chapters IV and V.

III. SYSTEM SOFTWARE

The Micropolis interfacing work consisted of two phases: the hardware interface design and the software interface. Before either could be accomplished, it was necessary to understand both the existing system architecture and software. The last chapter addressed that architecture. Therefore, this chapter presents the details of the AEGIS developmental system software.

A. CP/M-86 OPERATING SYSTEM

1. General Discussion

CP/M-86 is the operating system used in the AEGIS software development system. It is a commercially distributed operating system developed by Digital Research for use with a single INTEL 8086 based microcomputer. It is supplied on two single sided, single density, eight inch floppy disks. Included on these diskettes is the operating system (CPM.SYS), an 8086 assembler (ASM86.CMD), the Dynamic Machine Language Debugger (DDT86.CMD), an editor (ED.CMD) and various reconfiguration and file handling utilities.

The CP/M-86 operating system can be user configured to fit any hardware environment. As it is shipped, the file CPM.SYS is configured for 32K bytes of RAM, a keyboard, a screen device, an INTEL iSBC 204 Floppy Disk Controller and

a 9600 baud serially interfaced printer. The details of the CP/M structure and the reconfiguration procedures are discussed below. For information on the entire CP/M-86 environment and capabilities see [Ref. 10-12].

2. Structure

The CP/M operating system can be viewed as occupying three distinct levels of abstraction. Refer to Figure 3.1. Applications programs invoke system functions through the Basic Disk Operating System (BDOS) module and do not communicate with any other module. The BDOS performs services requested by applications programs and all general file and disk management functions. All hardware dependent functions required by the BDOS are requested through the Basic Input/Output System (BIOS) module. The BIOS module is the only one that communicates with the hardware. The Console Command Processor (CCP) shown is used to process console commands and provides the user interface in the absence of an applications program.

Since all hardware dependent functions are located in the BIOS module, the system hardware configuration must be reflected here. A skeletal BIOS (BIOS.A86) is provided in source code format for this purpose. The CCP and BDOS modules are provided as a single hex file (CPM.H86). This file requires no modification but is necessary for the adaption/reconfiguration process described in Section 4 below.

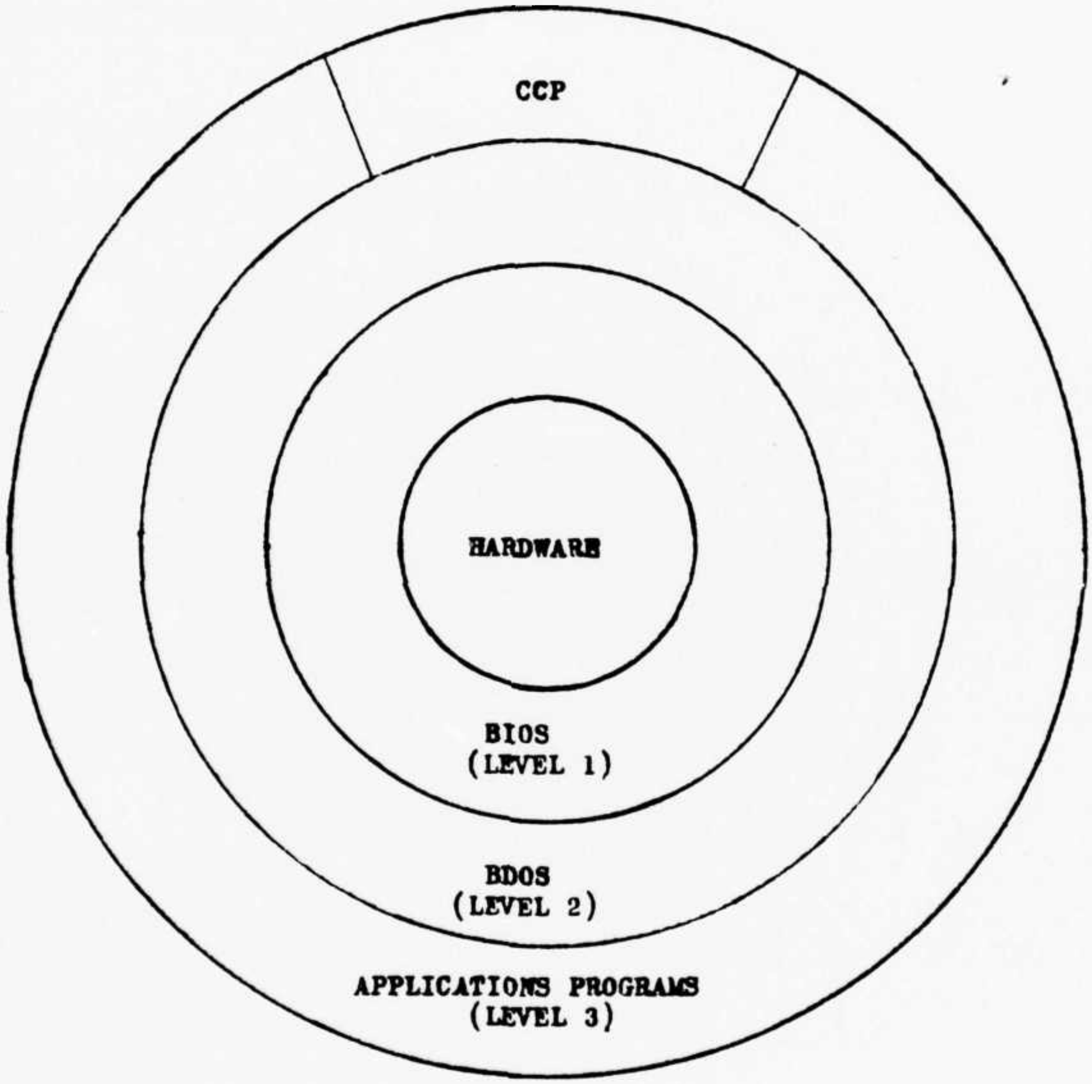


Figure 3.1 CP/M-86 Levels of Abstraction

3. Bootstrapping CP/M

Loading CP/M into RAM from a standard single density floppy disk requires a two step procedure. The boot ROM, which receives control when the system reset button is depressed, must load a loader program from the reserved system tracks on the disk into RAM and pass control to it. The loader is then responsible for loading the operating system from the disk into RAM and passing control to it. This two step procedure is required because the operating system is too large to fit on the reserved system tracks. Therefore, adaption of CP/M to a system other than that for which it is commercially distributed requires modification to these three software components.

4. General Adaption Procedures

The major effort in the adaption process is in the development of the hardware drivers for the BIOS module. The BIOS can be classified as performing three types of functions: hardware initialization/reinitialization, character I/O and disk I/O. The functions are contained in 21 subroutines within the module. The BIOS accesses the subroutines through a table that has individual jump vectors to the entry point of each subroutine. This is shown in the operating system memory map in Figure 3.2. The actions that must take place upon entry to each of these subroutines is detailed in [Ref 10: pp. 60-65]. A change in the hardware environment is accounted for by changing the code within

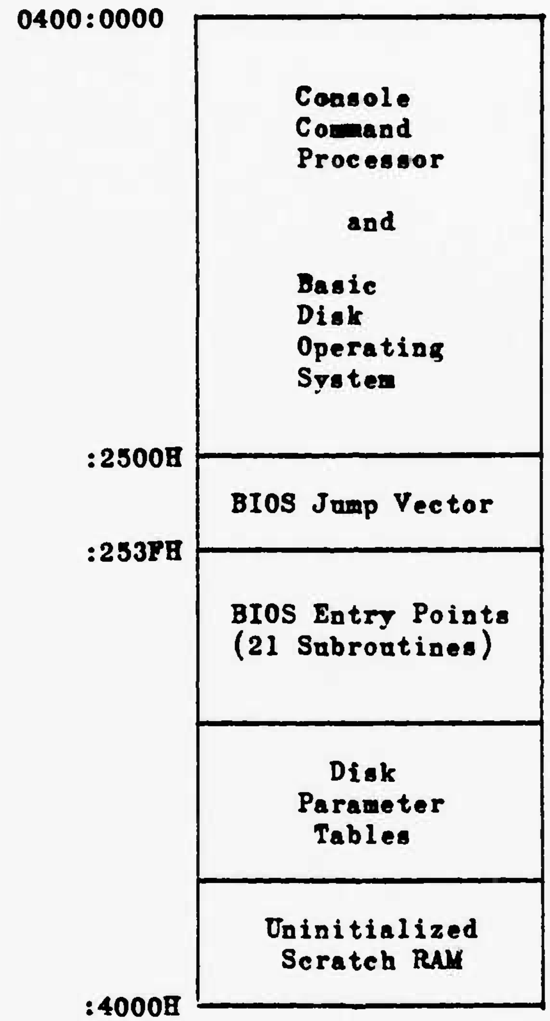


Figure 3.2 Operating System Memory Map

the 21 subroutines and meeting the entry and exit conditions as specified in this reference. Recall that a skeletal BIOS.A86 file is provided as a model for this purpose.

The Disk Parameter Tables shown in the previous figure are used by the BIOS to obtain the characterisctics of each device. These tables exist in a file separate from the BIOS and are included durin assembly through the use of the INCLUDE <filename>.<filetype> instruction at the base of the BIOS. The source code for the tables as well as the Uninitialized Scratch Ram Area, can be automatically generated by the GENDEF.CMD utility. This requires a <filename>.DEF file as input and produces a <filename>.LIB file as output. The contents of <filename>.DEF are simple, one line disk definition statements. The format for the statements and their meaning is described in detail in [Ref 10: pp.72-80].

. Once the BIOS file is modified and the Disk Parameter Table file created, they are assembled using ASM86.CMD. The resulting hex file is concatenated with CPM.H86 using PIP.CMD and a command file for this single hex file is generated using GENCMD.CMD. Finally, the new operating system that results is placed on the disk as CPM.SYS using PIP.CMD. The process described above is depicted in Figure 3.3. Note that the 8080 model option of CP/M-86 is shown in this example.

```
1.  USER.DEF ==> GENDEF.CMD ==> USER.LIB

2.  USERBIOS.A86 ==> ASM86.CMD ==> USERBIOS.H86

3.  CPM.H86 + USERBIOS.H86 ==> PIP.CMD
            ==> CPMSYS.H86

4.  CPMSYS.H86 ==> GENCMD.CMD ==> CPMSYS.CMD
            (8080 code[a40])

5.  CPMSYS.CMD ==> PIP.CMD ==> CPM.SYS
            (RENAME ON NEW DISK)
```

Figure 3.3 Steps for Creating a New CPM.SYS

Two software components remain to be adapted: the loader program and the boot ROM program. The loader program is a simplified version of CP/M-86 and contains only enough file processing capability to read the CPM.SYS file from disk to memory. Three files are provided for the development of a loader: LDCPM.H86, LDBDOS.H86 and a skeletal LDBIOS.A86 source file. The LDBIOS.A86 file reflects the hardware to be used in the loading operation and does not necessarily reflect the total hardware. This file contains the same 21 entry points as the BIOS.A86 file with the same entry and exit conditions and requires the same type of Disk Parameter Tables and scratch pad area. The generation of the LOADER.CMD file is depicted in Figure 3.4. The resulting loader must be small enough to fit entirely on the reserved system tracks.

1. URLDBIOS.A86 ==> ASM86.CMD ==> URLDBIOS.H86

2. LDCPM.H86 + LDBDOS.H86 + URLDBIOS.H86==>PIP.CMD
==> LOADER.H86

3. LOADER.H86 ==> GENCMD.CMD ==> LOADER.CMD

4. LOADER.CMD ==> LDCOPY.CMD ==> LOADER.CMD
(LOAD ON SYSTEM TRACKS)

Figure 3.4 Steps for Creating LOADER.CMD

The development of a boot ROM program depends only on the physical device used to load the operating system. Its single purpose is to load the program located on the system tracks into RAM and pass control to it. A ROM.A86 file is provided that details the boot ROM for an INTEL iSBC 204 Floppy Disk Controller and serves as an excellent example. However, because the method used will vary widely from device to device, no files are provided that simplify this development.

B. AEGIS IMPLEMENTATION OF CP/M-86

1. Boot ROM and Loader

In the AEGIS implementation of CP/M-86 used during the initial development work of this thesis, two boot ROM programs and their associated loader programs were available. Both are located at the base of the INTEL 957 monitor in the 8K byte EPROM of the iSBC 86/12A. The first

allows the system to be booted from either the single or double density INTEL MDS floppy disk drive by executing the command "GFFD4:0" from the monitor. The second will boot the system from the bubble memory by executing "GFFD4:4" from the monitor.

2. A Modification to the BIOS

Recall from Section A-4 above that any hardware change within the system requires some of the 21 BIOS subroutines to be rewritten. A change occurs not only by the addition of hardware but also when a component is removed either because it has failed and there is no replacement or it is no longer needed. In a hardware environment as flexible as that required by the AEGIS project, the standard reconfiguration process becomes an extremely time-consuming task.

To alleviate this problem, a method was developed as a part of Almquist and Stevens' work in which only minor changes to the BIOS were required to alter the hardware configuration. With this technique, all of the BIOS device-dependent subroutines are extracted into a separate file for each unique device. The specific device-dependent routines are: INIT, SELDSK, HOME, SETTRK, SETSEC, READ and WRITE. The physical location of the entry points to the routines is obtained from an ordered label table file and the BIOS accesses the routines through an indexed CALL instruction.

How this technique works is shown in Figure 3.5. In this example, a call is made to the BIOS to READ DEVICE #2. The BIOS makes a jump to the READ entry point. However, instead of doing an actual READ at this point, the device number is determined and a call to the address found in the second position of RDTBL is made. The code that performs the READ function for DEVICE #2 is then executed.

The code for the seven device-dependent functions can be written and debugged independently of any other code. To add the device to the system requires one INCLUDE <filename>.<filetype> statement be added to the BIOS file, the corresponding seven entry points be added to the label table file and the Disk Parameter Table be updated. The steps for creating the CPM.SYS file remain unchanged from those presented in Section A-4. To remove a device, the process is reversed. Clearly this method allows the hardware dependent code (and hence, the hareware itself) to be more easily integrated in or removed from the operating system than the standard BIOS structure did.

C. MULTIUSER SYSTEM

CP/M is not a multiuser or multitasking operating system. Another major development of the Almquist and Stevens' research work was a method by which each single board computer in the system could operate independently of the others under CP/M and still have access to the shared

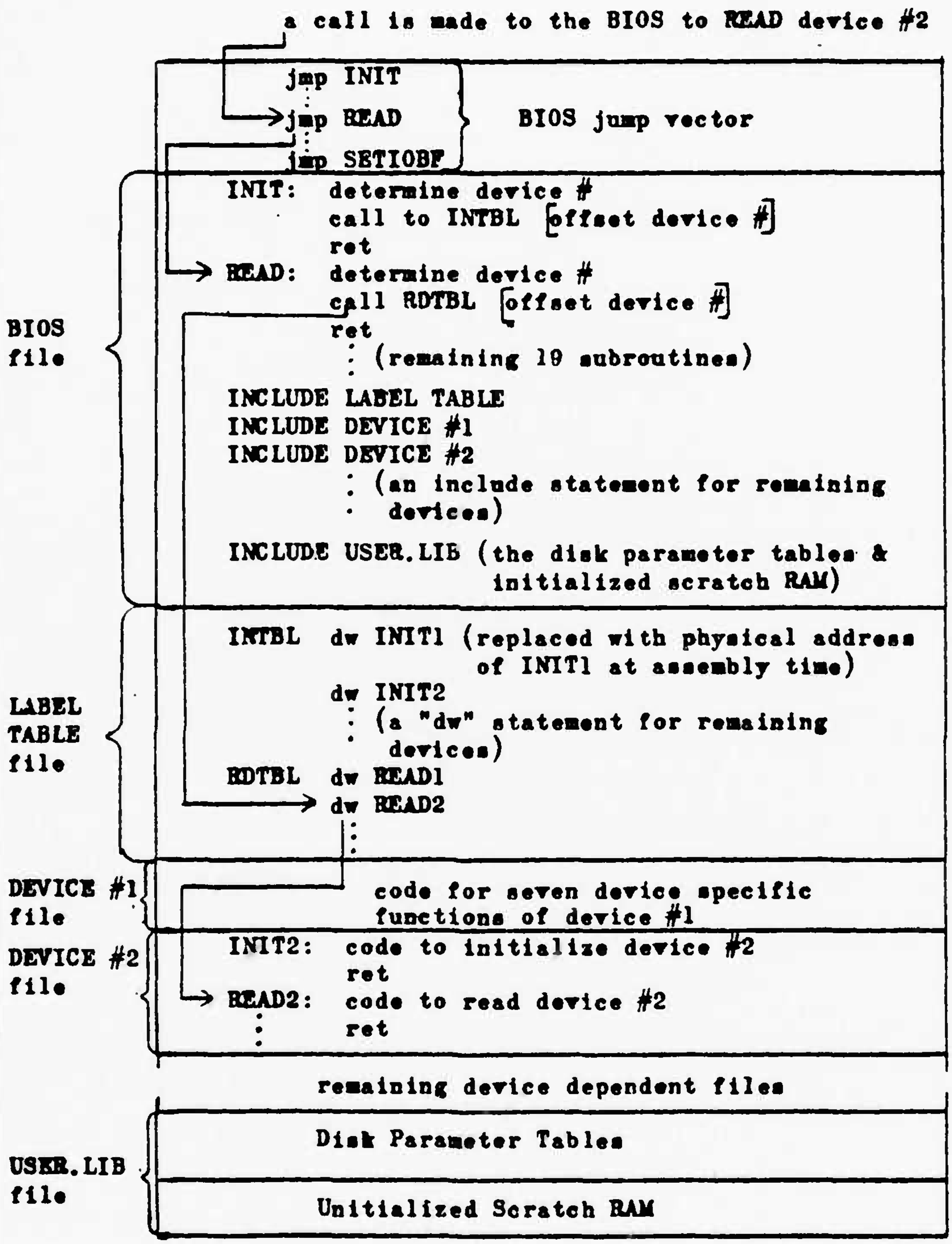


Figure 3.5 Path of a BIOS Function Call

resources (the disk drives and the bubble memory) of the system. The multiuser system that resulted can be broken down into three functional categories: syncnronization of common memory usage, boot loading all SBC's and write protection of a user's allocated disk space.

1. Syncnronization

As stated in Chapter II, the RAM on each iSBC 86/12A is not accessible via the MULTIBUS and therefore, all disk and bubble memory read/write operations must be performed through a buffer in the common memory. This requires a syncnronization scheme that will ensure a single computer can successfully complete a read/write operation before another computer is permitted access.

A ticket/server technique had been developed for this purpose. This required a CALL REQUEST instruction to be placed prior to all common memory read/write operations and a CALL RELEASE instruction be placed after the read/write operation. The CALL REQUEST accesses the "ticket" variable in common memory for a ticket number and waits until that number is equal to the "server" variable, a number also found in common memory. The read/write operation is then performed and the CALL RELEASE advances the server number, thus releasing common memory to the next ticket holder.

The code for these subroutines is contained in the file SYNC.A86. It is included in the BIOS through an

INCLUDE statement placed immediately following the last INCLUDE statement for the device files.

2. Boot Loading All iSBC's

Because the common memory variables must be initialized only once, two versions of the CP/M-86 operating system had been developed. The file CPMMAST.CMD is the master version that performs the common memory initialization while CPMSLAVE.CMD is the slave version that does not.

The master board is boot loaded with CPMMAST.CMD from the bubble memory. Once this board is operational, the command "LDBOOT" is executed and results in a copy of BOOT.CMD being placed in the common memory. Next, the command "LDCPM" is executed. This places a copy of CPMSLAVE.CMD into common memory. From the monitor of the remaining SBC's, the command "GE000:0400" is executed. This causes the CPU to execute the code of BOOT.CMD which, in turn, moves a copy of CPMSLAVE.CMD into local RAM and transfers control to it.

3. Disk Write Protection

Disk write protection was achieved through a log-in procedure. A log table is initialized in common memory as a part of the master version initialization. The number of entries in the table correspond to the number of disk drives or logical devices in the system. As part of booting the

operating system, the user is queried for the console number being used (located on the front panel of each console) and the disk drive to log on to. The log table is checked after this entry to determine if the desired drive is free. If it is, the user console number is placed at that drive's position in the log table. If it is not, the user is asked to re-select a drive.

The user console number is also stored in the local variable USER. Write protection is accomplished by comparing the value in USER to the currently selected logical disk number and aborting any write operation if they are not the same.

It was considered desirable to be able to change disk drives without the need to reboot the board. A log out procedure was written for this purpose. When the command "LOGOUT" is executed, the USER variable is reset, the log table is updated and the user is again requested to enter a disk drive to log on to.

The final common memory utilization employed as a result of the multiuser system developed by Almquist and Stevens is depicted in Figure 3.6.

D. MCORTEX

MCORTEX is the operating system that was developed for the SPY-1A radar emulation. In the final version presented by Klinefelter [Ref. 2], MCORTEX was set up to handle ten

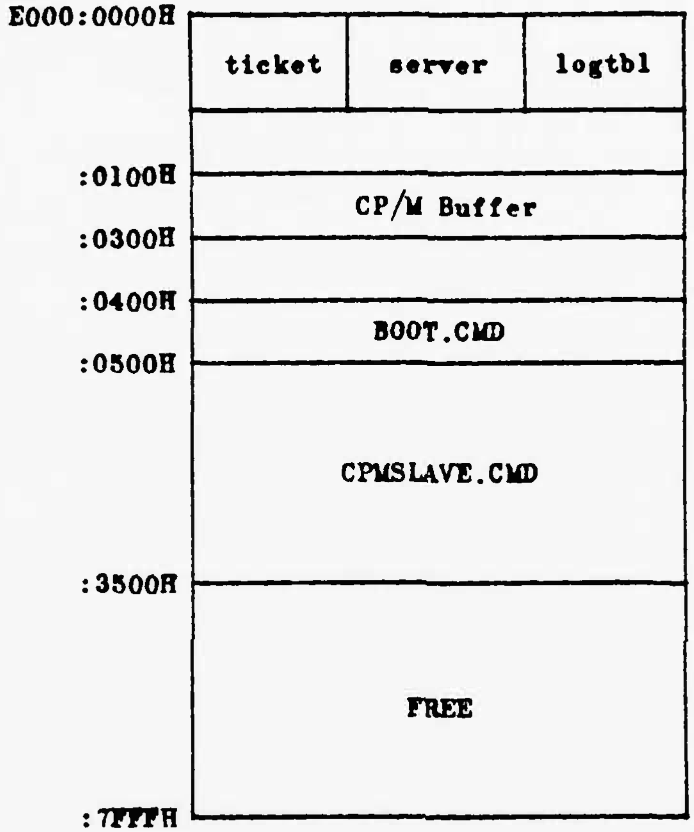


Figure 3.6 Common Memory Allocation Map

virtual processors for each real processor. The data base upon which all scheduling decisions are based is the Virtual Processor Map (VPM) located in common memory. The VPM contains the control and status information on each virtual processor required by MCORTEX to coordinate the concurrent processing.

All processes managed by MCORTEX can be in one of three states: running on a real processor, waiting for some event to occur or ready to run (waiting to gain access to the real processor). If a process is in the waiting state, it could be waiting for an event to occur on a real processor other than the one to which it was assigned. An eventcount table is maintained in common memory for notification purposes. Whenever a real processor completes an event, the table is updated and a message is broadcast to all other real processors in the system that some event has just been completed. Each real processor then checks the eventcount table to determine if the event pertains to any of its virtual processors and reacts accordingly.

The technique used for broadcasting an event employed a type of global interrupt issued on the MULTIBUS. Because the development of the Micropolis interface altered the technique somewhat, the details of it will be presented in the next chapter.

IV. MICROPOLIS HARDWARE INTERFACE DEVELOPMENT

A. OVERVIEW

The Micropolis disk system offers an interface structure that is suitable for use with either programmed I/O or DMA data transfers. In an effort to make maximum use of available system hardware, the programmed I/O mode was used in this implementation. This enabled an interface to be designed with the INTEL 8255 programmable parallel I/O chip located on the iSBC 86/12A.

Recall from Chapter II that all RAM on each SBC is not accessible through the MULTIBUS. Therefore, all control, status, and data information intended for the Micropolis disk system had to be passed through the common memory. Because the disk system was interfaced into a single SBC's 8255 port, the disk controller had no method of communicating directly with the common memory. To overcome this problem, a timer-controlled interrupt was designed. This allowed the 8086 CPU to be interrupted at periodic intervals and effect any necessary communications between common memory and the controller. The distinct advantage of this technique is that the single board computer used for the interfacing can still be used as an independently operating computer with all disk operations being transparent to the user.

In the following paragraphs, the details of the interface design and the timer-controlled interrupt design are presented. This discussion is limited to the development of the designs and only those low-level routines needed to test their validity. The software implementation into the AEGIS system will be discussed in Chapter V.

B. MICROPOLIS DISK SYSTEM

Chapter II stated the general characteristics of the Micropolis disk system. This section expands on that by presenting the technical interface requirements as well as the general operation of the disk controller. For more information on the Micropolis disk unit see [Ref. 13].

1. Interface Signals

Interface to the Micropolis disk drive is made through a 34 pin edge connector located on the controller printed circuit board. The interface is structured around an 8-bit bi-directional data bus and 9 control lines. For ease of reference, the 8 data lines will hereafter be referred to as BUS0-BUS7 with the BUS0 line corresponding to the least significant bit and the BUS7 line the most significant bit. The control line names and a complete description of each is contained in the list below. Note that in this list, reference is made to the logical condition of the signal (true = 1 and false = 0) rather than the signal's electrical polarity.

a. SEL: Since the Micropolis controller can slave another disk unit off of it, this signal is provided to select which disk unit to use. This application only utilizes one disk unit and it is jumper configured to respond to address Ø. Thus, SEL should always be Ø.

b. ENABLE: This signal is normally held true. If made false (2 microseconds minimum), a reset is applied to the controller logic. The controller will indicate that it is busy (through the CBUSY signal described below), important flags and registers are then initialized and approximately one second later, the controller will indicate that it is ready to accept commands from the host computer.

c. WSTR: The write strobe is a signal from the host computer to the controller that the information on BUSØ-BUS7 is valid. The host pulses the write strobe line while driving the bus. On the trailing edge of WSTR, the controller will copy the contents of the bus into a buffer. The byte is then interpreted by the controller as either control (DATA = Ø) or data (DATA = 1).

d. RSTR: The read strobe is a signal used by the host to indicate to the controller that it is ready to input a byte of information. When the host drives RSTR true, the controller drives the bus with the contents of either its data buffer (DATA = 1) or its status register (DATA = Ø). The controller will drive the bus as long as RSTR is true.

Thus, once the host has copied the bus, RSTR must be made false again to regain access to the bus.

e. DATA: This signal selects either the controller data or control ports as described above.

f. CBUSY: The controller will set CBUSY to 0 whenever the host issues it a command. CBUSY is returned to 1 by the controller when the command is terminated.

g. ATTN: The attention signal is set true by the controller at the end of each command. The host responds by reading the Termination Status byte from the data port. ATTN is set false by the controller only after the Termination Status byte has been read.

h. DREQ: Data request is used to request the transfer of data to/from the controller. The direction of the transfer is controlled by the OUT signal. Data can only be transferred by the host in response to DREQ.

i. OUT: This indicates the direction of data transfer. If OUT = 1, the transfer is from controller to host (a READ operation). If OUT = 0, it is from host to controller (a WRITE operation).

All bus lines and control signals (except CBUSY) are active low at the interface connector. The physical interface to the 8 controller bus lines must be through an INTEL 8226 inverting bi-directional driver/receiver or its equivalent provided by the host system. The host must also provide 1K ohm pullup resistors on each of the bus lines.

Interface to the SEL, ENABLE, DATA, WSTR, and RSTR control lines is through a 7438 inverting driver or its equivalent. The ATTN, CBUSY, DREQ, and OUT control signals are used in a DMA interface environment. If operation is in the programmed I/O mode, the DMA signals do not have to be physically connected. The logical condition of these signals can be obtained by reading the status register (see RSTR above).

2. <u>General Operation</u>

Command execution is started in the controller board by writing a command byte to the command port, followed by writing six parameter bytes and a GO byte to the data port. The command byte identifies the type of command while the parameter bytes contain the control and addressing information necessary to execute it. The GO byte actually starts the command execution and can contain any value. After the controller has executed the command, a Termination Status byte is written to its data port and ATTN is set. When the host reads this byte, the command execution is complete and the controller can accept a new command.

3. <u>Commands and Error Recovery</u>

Three types of commands can be executed: non-data transfer, transfers from host to controller (write), and transfers from controller to host (read). The non-data transfer commands are used for disk maintenance. This set of commands permit, among other things, the initialization

and/or verification of all 580 tracks associated with one of the five drive heads.

The read and write commands have three major options: operation on an entire track or a single sector, programmed I/O mode or DMA mode, and automatic retries enabled or disabled. The single sector, programmed I/O, and automatic retries enabled options were used for all read/write operations in this implementation of Micropolis. The automatic retries feature is an extremely powerful one and warrants further discussion.

If automatic retries are enabled, three levels of retry are performed by the disk controller for data errors. In level one, a correction attempt is made on the data using the standard CRC-CCITT 16th order polynomial. If the correction attempt was successful, the corrected data is transmitted to the host. If not successful, a level two retry is invoked. Level two will repeat the operation and correction attempt up to ten times. If still unsucessful, a level three retry begins. In level three, the read amplifier gain is increased and level one and two retries are performed. If this fails, the head positioner is offset one way then the other from the track center and level one and two retries are performed again. If all retries fail, the command is aborted and an error condition is placed in the Termination Status byte. This feature clearly provides for a high degree of data integrity and error recovery.

4. <u>Parameters</u>

All six parameter bytes required as a part of command execution must be transmitted to the controller even though some may not be used. A description of those parameters is contained in the list below.

a. Parm 1: Bits 4-7 contain the head address (a value between Ø and 4). Bits 2 and 3 are set to Ø and bits 1 and 2 contain the unit address. Recall that only one Micropolis disk unit is used and that its address is Ø. This makes only five values valid for Parm 1 depending on the head selected: ØØH, 1ØH, 2ØH, 3ØH, and 4ØH.

b. Parm 2: This parameter contains the least significant 8 bits of the track address.

c. Parm 3: Bits Ø-2 contain the most significant 3 bits of the track address and all others are set to Ø.

d. Parm 4: Contains the sector address (a value between Ø and 23).

e. Parm 5: Contains the number of sectors to process. In this implementation, this value is set to 1.

f. Parm 6: This parameter is used only in track oriented commands and since sector read/write operations were used, this byte is always set to Ø.

C. PREVIOUS WORK

The Micropolis unit had been previously interfaced in an INTEL MDS single user environment by James John [Ref. 14].

This interface was constructed using the iSBC's onboard 8255 programmable I/O device.

The 8255 can be configured in a combination of three modes: mode Ø, mode 1, and mode 2. These modes and the operation of the device is discussed in detail in [Ref. 15]. In John's application, the 8255 was programmed in mode 2 and mode Ø. Mode 2 provided 8 bi-directional data lines at Port A and 5 control lines for the bi-directional data port and 3 output only lines at Port C. Mode Ø provided 8 output only lines at Port B.

The required 8226 drivers with 1K ohm pullup resistors are hard-wired on the 86/12A in line with the bi-directional data port of the 8255 and did not have to be added. The required 7438 drivers were inserted in sockets A11 and A12 on the iSBC in line with Ports B and C. John's final interface design is depicted in Figure 4.1. (All active low signals are indicated by following the signal name with a "/", such as ACK/. This notation will be used throughout th remainder of this thesis.) Note from this figure that the STB/ and ACK/ signals needed by the 8255 to latch input data and enable the tri-state output buffer are provided by wiring two of the Port B output lines into the STB/ and ACK/ inputs. These signals must be controlled locally as the disk controller provides no compatible signals.

As part of James John's work, he also reconfigured the CP/M BIOS to accomodate the Micropolis disk and two INTEL

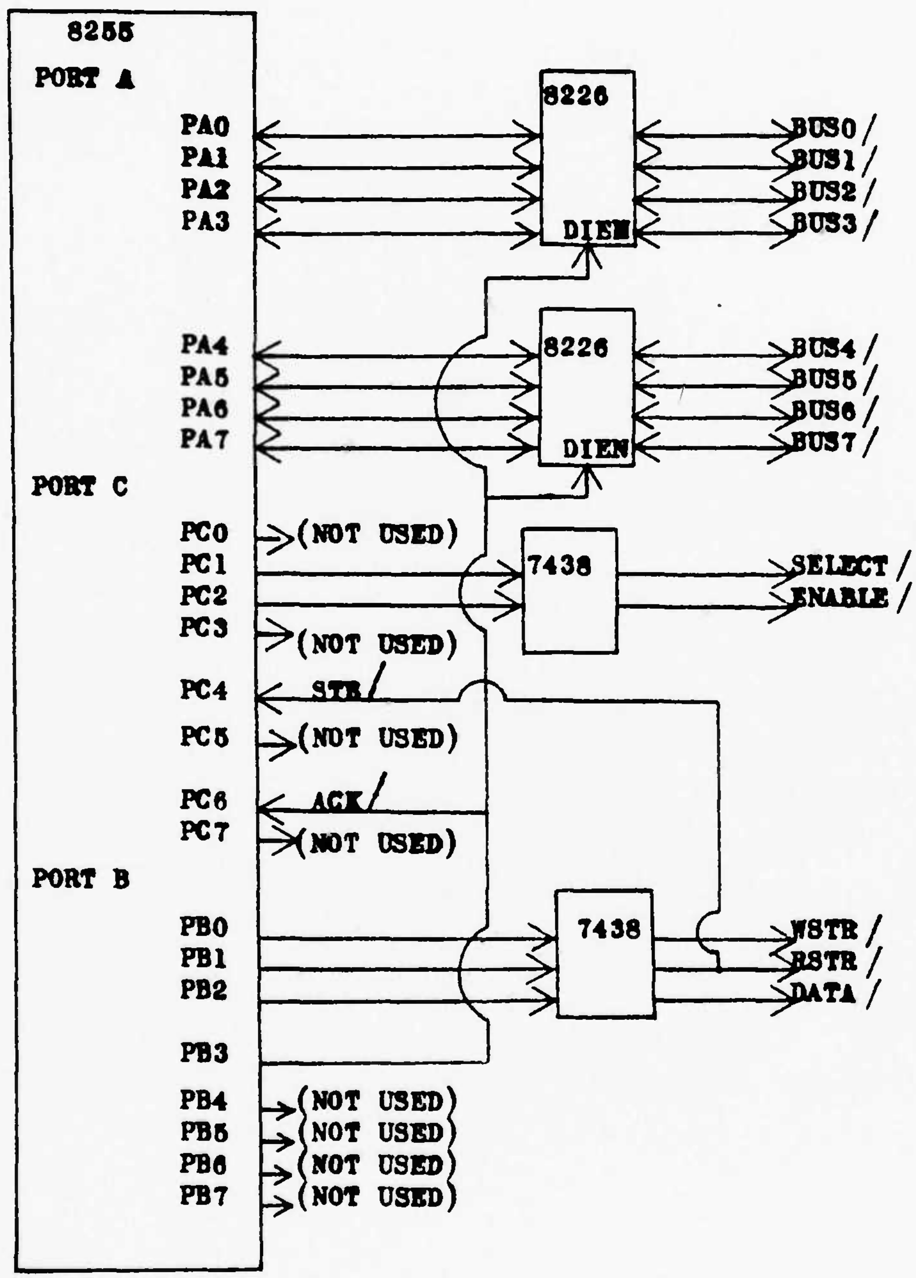


Figure 4.1 Previous Design

MDS floppy disk units. This gave a complete, single user system with a total of seven accessible drives.

D. INITIAL EFFORTS

It was envisioned to expand on John's work to accomplish integration of the Micropolis disk drive into the AEGIS multiuser development system. Therefore, the first logical step was to set up James John's system and test it.

Various files were read from the floppy drives and written to the Micropolis drives and vice versa. This orginally appeared successful. However, whenever a source code assembly language file was read and an assembly attempted on that file, the assembly continuously failed. A print-out of the source code file was obtained and various errors were found that did not exist in the original file. This led to the belief that a bad copy of the ASM86.CMD assembler was being used and it was crashing not only the system but also the file it was operating on. A good copy of the assembler was obtained and the test repeated with continued negative results. Hardware connections were verified and re-verified. Software was also checked and re-checked. Nevertheless, numerous other experiments still produced negative results.

At this point, the design was re-examined and this uncovered the problem. A timing analysis was performed and is presented in Figure 4.2. The latch of WRITE data from

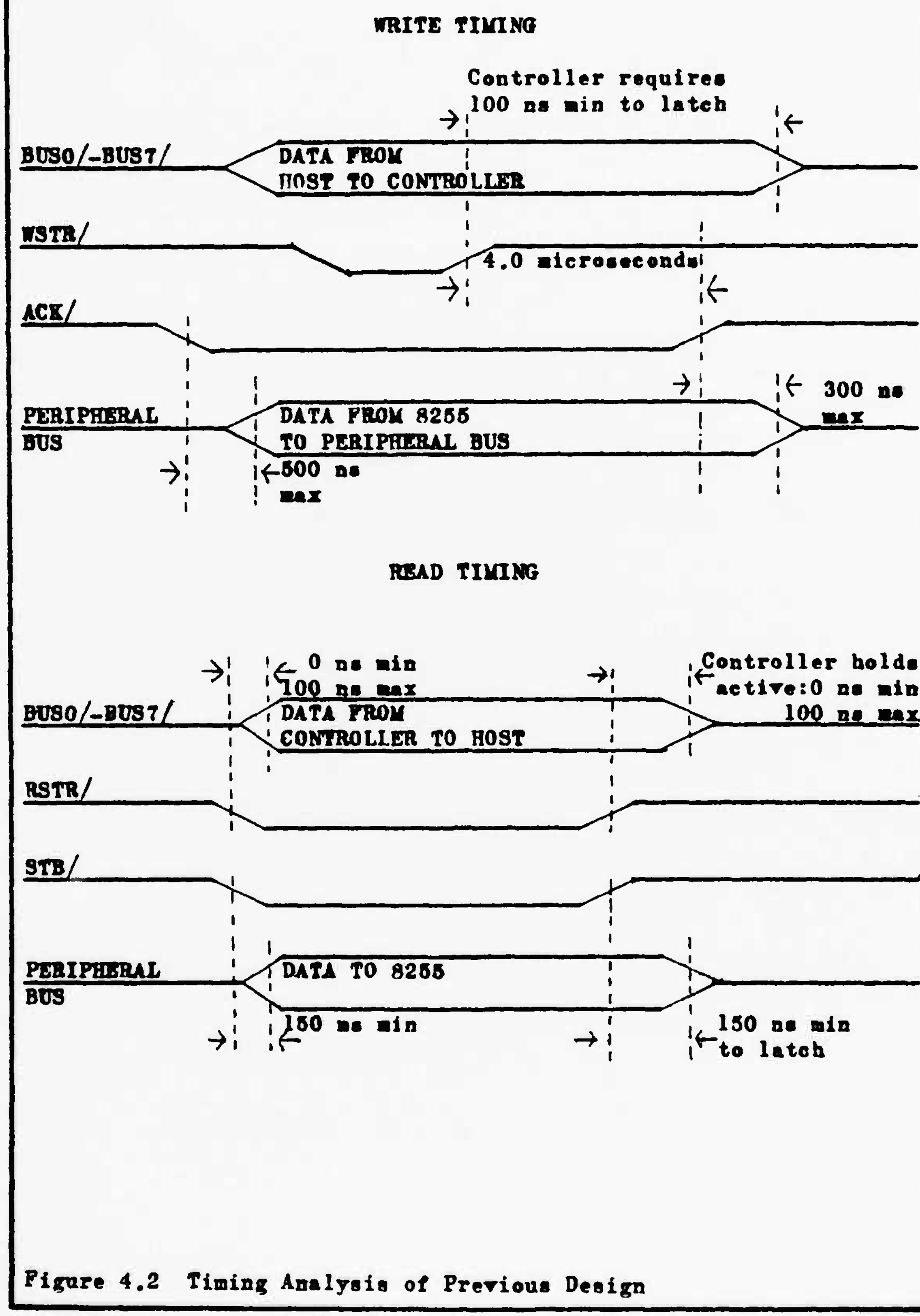


Figure 4.2 Timing Analysis of Previous Design

the host to the controller occurs within a valid region (window) as shown. However, the latch of READ data from the controller to the host does not. This occurs because the RSTR/ and STB/ signals are physically wired together. Both will go active/inactive at exactly the same time. The controller will hold the bus active only for 100 nanoseconds maximum after RSTR/ goes inactive. However, the 8255 requires 150 nanoseconds minimum to latch the data after STB/ goes inactive. Thus, the data that is latched may or may not be valid. This explains why marginal success was obtained when source files were just written to and read from the Micropolis. It also explains why random errors that were not present in the original source file were found in the file that was printed from the Micropolis.

E. NEW DEVELOPMENT

1. Design

A new interface was designed in which the ACK/, STB/, RSTR/, and WSTR/ signals were all independently controlled by setting the appropriate bit on the 8255. Because the condition of each individual signal is now under software control, it can be ensured that the data will remain valid long enough for either the controller or the 8255 to latch it. The new design is presented in Figure 4.3 and the associated timing diagram in Figure 4.4. As shown in Figure 4.4, all latching occurs in valid windows.

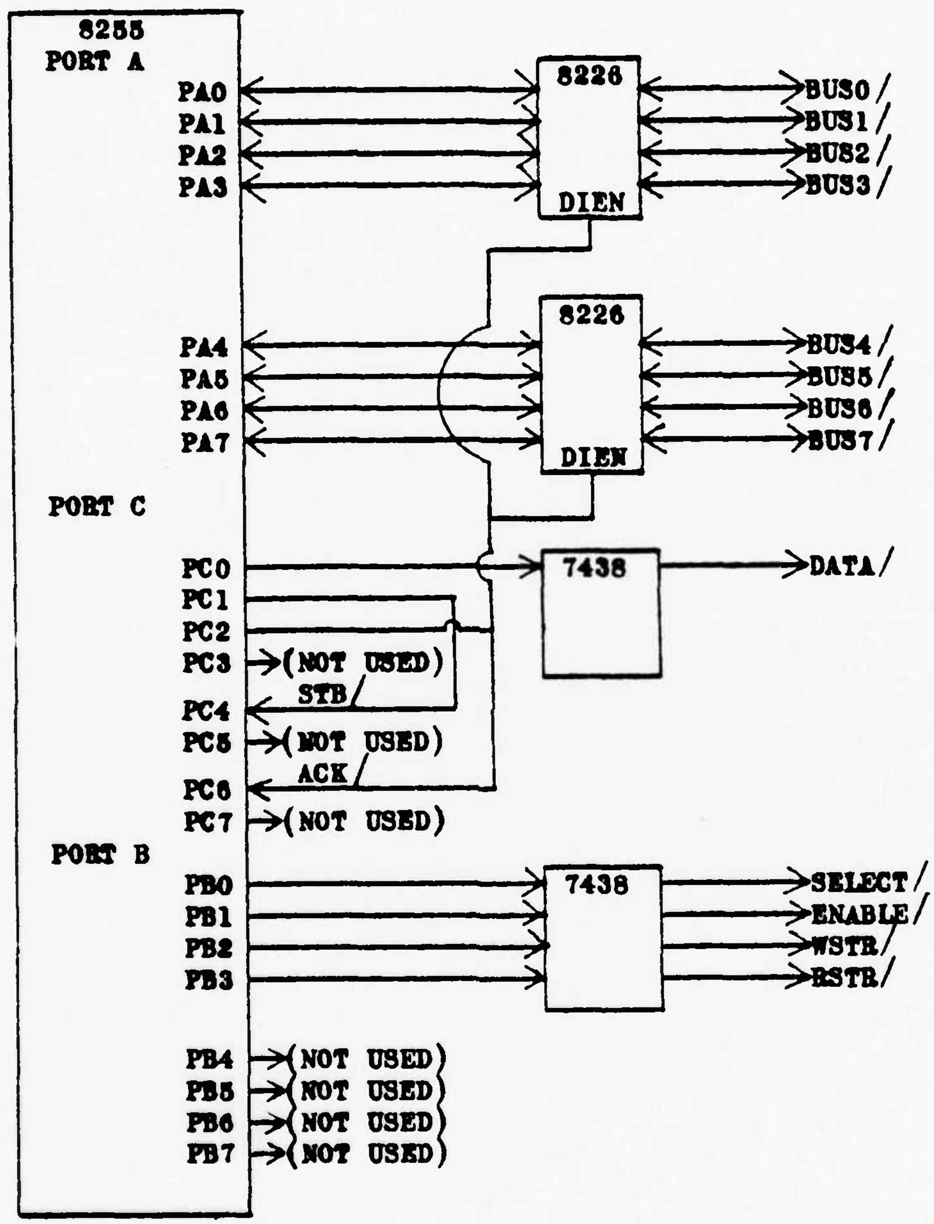


Figure 4.3 Interface Design

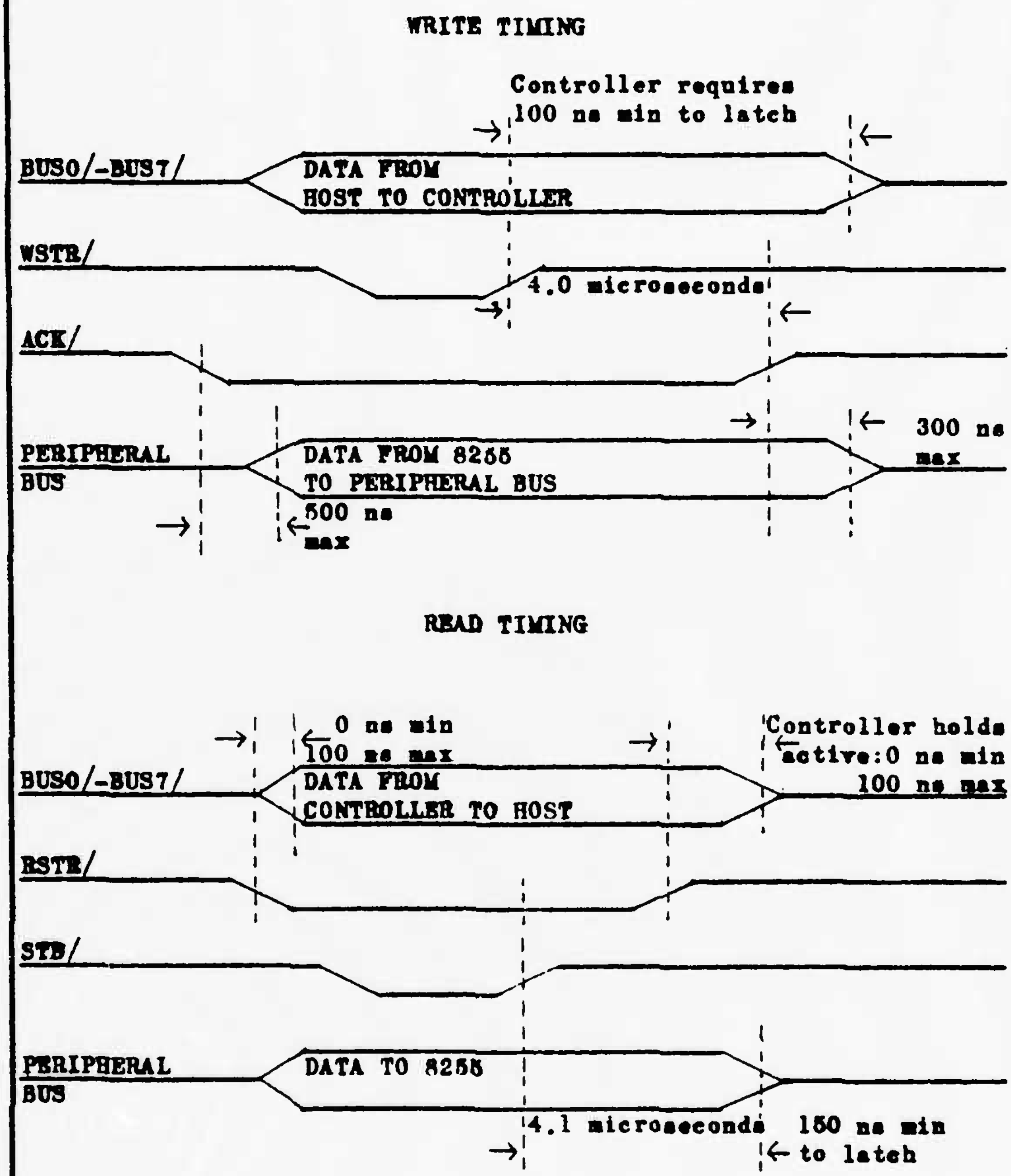


Figure 4.4 Timing Analysis of Interface Design

To write a byte of information to the controller, the byte is first written to Port A of the 8255 and the ACK/ signal is set to 0, enabling the tri-state output buffer and driving the bus lines with the contents of that buffer. Next, WSTR/ is turned on (set to 1) and turned off (set to 0). The controller will copy the bus lines into an input buffer on the trailing edge of WSTR/. Finally, the tri-state buffer of the 8255 is disabled by setting ACK/ to 1 and the write operation is complete.

To read a byte of information from the controller, RSTR/ is activated. This will cause the controller to drive the bus lines with its data buffer as long as RSTR/ remains active. Next, STB/ is turned on (set to 0) and turned off (set to 1). This copies the bus lines into the 8255 input buffer. Lastly, the RSTR/ signal is deactivated and the byte can be accessed by reading Port A of the 8255.

Each of the bit set/reset operations needed in the read or write functions just described, requires a MOV and an OUT instruction for a total of 20 clock periods in the 8086. With a 5 Mhz clock, this is 4.0 microseconds and clearly, more than meets the response or settling time constraints of either the 8255 or the disk controller.

2. Implementation and Testing

The design in Figure 4.3 was set up on an iSBC 86/12A. The following hardware changes were required to the board:

Remove jumpers:

E13-E14	E17-E18	E21-E25
E15-E16	E19-E20	E28-E29

Add jumpers:

E28-E15
E30-E17
E30-E25

Add 7438 drivers in sockets:

XA11
XA12

Next, a cable was constructed that would interface the J1 34 pin edge connector of the 86/12A to the J121 34 pin edge connector of the Micropolis controller. The cabling requirements are shown in Table 4.1. Those pins not shown are not required and are not connected.

The 86/12A was then placed in the iCS 80 chassis for testing. Only the most primitive routines were written to read and write to various heads, tracks, and sectors of the Micropolis. These were executed under DDT86 to allow manual changing of the command and parameter bytes. First, a single character was written to fill an entire sector and then read back. This was successful. Next, a text message was prepared and written to a variety of different sectors and tracks of each drive head. In each case, the message was retrieved successfully and it was concluded that the design was functional.

iSBC 86/12A J1 CONNECTOR PINS		DESCRIPTION	MICROPOLIS J101 CONNECTOR PINS	
SIG	GND		SIG	GND
48	(NONE)	← BIT 0 →	16	(NONE)
46	45	← BIT 1 →	14	13
44	43	← BIT 2 →	12	11
42	41	← BIT 3 →	10	9
40	39	← BIT 4 →	8	7
38	37	← BIT 5 →	6	5
36	35	← BIT 6 →	4	3
34	33	← BIT 7 →	2	1
24	23	← DATA →	20	19
16	15	← SEL →	28	27
14	13	← ENABLE →	26	25
12	11	← WSTR →	24	23
10	9	← RSTR →	22	21

Table 4.1 Interface Cable Connection Requirements

F. INTERRUPT MECHANISM

1. Design

With the interface design complete, it remained to design the timer-controlled interrupt for polling common memory. The design was based on two devices available on the 86/12A: the INTEL 8253 programmable interval timer and the INTEL 8259 peripheral interrupt controller.

The 8253 has three independent 16-bit counters and each can be programmed in one of five modes. Details of its operation can be found in [Ref. 15]. The design employed here uses only counter Ø and it is programmed in mode Ø, the "interrupt on terminal count" mode. In this mode, the output of the timer will be driven low when the mode control word is written to it. After the count value is loaded into the count port, the counter will begin counting down. Upon reaching the terminal count, the timer output will go high and remain high until a new count value is loaded.

The mode control word selected was 30H. This gives timer Ø the following characteristics: operation in mode Ø, binary 16-bit counter, and load count value as least significant byte first then most significant byte. The count value used was 300CH which corresponds to an interval of 10 milliseconds at a clock frequency of 1.23 MHZ (the clock frequency supplied to the 8253 by factory default).

Like the 8253, the INTEL 8259 has many different options available. Only those appropriate to this design

are covered in the following paragraphs. For more information see [Ref. 15].

Three initialization command words (ICW) and one operational control word (OCW) are required to properly configure the 8259. ICW1 is set to 13H. This corresponds to edge triggering, no slave interrupt controllers, and ICW3 is not required.

ICW2 is set to 40H. This is used in conjunction with the interrupt level number to arrive at the address in the interrupt vector table (see Figure 2.3) from which to obtain the code segment and offset values for the interrupt handler routine. Interrupt level 6 was chosen and this corresponds to a vector table address of:

4 * (40H + 6H) = 118H

Therefore, the address of the interrupt handler must be loaded in this location.

ICW4 is set to 0FH. This indicates 8086 mode, automatic end of interrupt, and buffered mode.

OCW1 is used to mask unused interrupts. It is set to BFH. This enables interrupt level 6 and disables all others.

2. <u>Implementation and Testing</u>

To implement the design simply required removing default jumper E79-E83 and connecting a jumper between E75 and E83. This connects the output of timer 0 on the 8253 to the interrupt 6 input on the 8259.

A primitive routine was written that initialized both devices as descibed above and loaded an interrupt handler address into the vector table. The interrupt handler was a simple routine that performed the following: saved all registers on the stack, printed a message at the console, restored all registers, and reloaded the count value into the timer. When tested, the timer-controlled interrupt functioned normally.

G. MCORTEX INTERRUPT

As stated in Chapter III, the MCORTEX operating system uses a type of global interrupt for message broadcasting. The hardware configuration required to achieve this is depicted in Figure 4.5. Port C of the 8255 is initialized as an output port and to "issue" the interrupt requires setting bit 7 of port C to 1.

It is envisioned that future development will allow the CP/M-86 operating system and MCORTEX to co-exist in the local RAM of each independent user on the AEGIS multiuser development system. This was taken into account in this research effort. Nevertheless, changes in the hardware and hardware initialization will be necessary before this can be achieved. Those changes are identified below.

Both MCORTEX and CP/M-86 (with the Micropolis integration), initialize the 8259 interrupt controller with exactly the same initialization command words. Interrupt

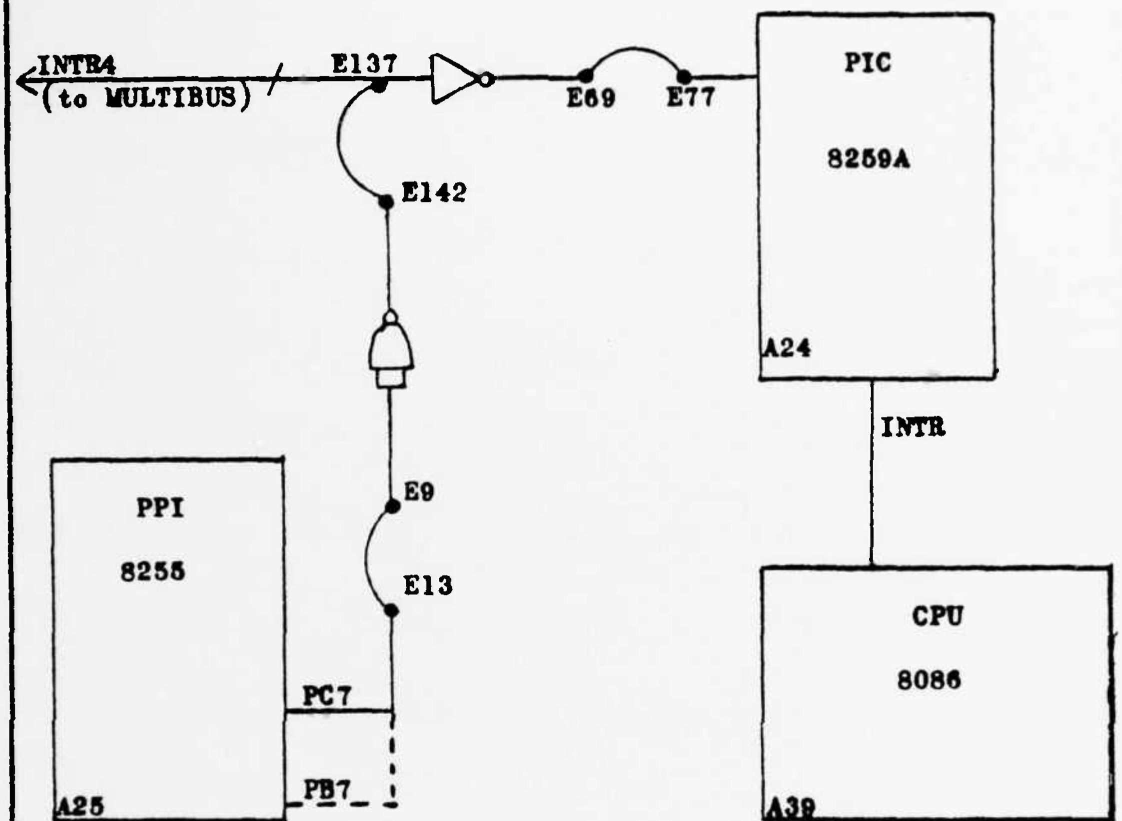


Figure 4.5 MCORTEX Hardware Interrupt Connections

level 6 was chosen for Micropolis to be of a lower priority than the interrupt level 4 used by MCORTEX. However, recall from Section F-1 that the operational command word is set to BFH and this masks all interrupts except level 6. This value will have to be changed to AFH to activate interrupt level 4.

The MCORTEX interrupt bit will also have to be moved as shown by the dashed line in Figure 4.5. This is required because the Micropolis disk drive uses Port C as a strobed input/output port. The hardware dependent source code found in the advance, pre-empt, and initialization processes of MCORTEX will have to be updated to reflect this change.

V. SOFTWARE IMPLEMENTATION

A. MAINTENANCE SOFTWARE

Before the Micropolis disk unit could be used, it was first necessary to write a routine that would initialize and format the disk surfaces. The purpose of initialization is to write the address and data fields of each sector onto the surface. This is a controller invoked function. After initialization, the address field will contain the required head number, track number, and sector number. All data fields will contain 51H.

The purpose of formatting is to change the contents of the data fields from the 51H that resulted during controller initialization to E5H. This is necessary because CP/M-86 expects to find E5H in the data fields in order to create a directory space.

The routine that was developed, MICMAINT.CMD, is completely menu driven and extensive error checking is performed on all user supplied input. This routine provides not only initialization and formatting options but also verification of initialization and verification of formatting. These additional facilities enable the user to easily uncover any disk surface defects. For an explanation of how to use MICMAINT.CMD, see APPENDIX A (User's Manual for the AEGIS System).

The Micropolis disk surfaces were successfully initialized and formatted with MICMAINT.CMD. No surface defects were found in the initialization process.

B. DEVELOPMENT OF THE DEVICE DEPENDENT ROUTINES

As stated in Chapter III, seven device dependent routines were necessary in order to interface the Micropolis disk drive into the AEGIS development system. The SELDSK, HOME, SETTRK, SETSEC, READ, and WRITE routines were developed simultaneously. This was a consequence of the Micropolis 512 byte physical sector length. The CP/M-86 operating system utilizes a 128 byte logical sector length. Therefore, a physical to logical sector mapping (blocking/deblocking technique) was required in order to communicate with CP/M. The method used had an effect on all six of these routines.

The INIT routine required special attention as it was used not only to initialize the parallel I/O port, the timer, and the interrupt controller but also to embed the the interrupt handler within the operating system. The details of both the INIT routine development and the blocking/deblocking algorithms used are given below.

1. Initialization and Interrupt Handler

The hardware initialization required for the INIT routine had been previously developed and tested (see Chapter IV). It remained to develop an interrupt handler.

Recall that the sole purpose of the interrupt handler is to effect communications between the Micropolis disk controller and the common memory. A status byte, command packet, and a 512 byte sector buffer were established in the common memory to coordinate this effort. Figure 5.1 depicts the resulting map of common memory addresses.

The status byte serves two purposes: to inform the interrupt handler that a disk read/write operation is being requested and to return the success/failure code that resulted during that operation. It is initialized to ØFFH as a part of the Micropolis INIT routine. The status byte is set to ØØH to request a disk operation and the interrupt handler will return ØAH if the operation was successful. If it failed, one of the nine error codes listed in the Micropolis Technical Manual pages 24-25 is returned.

The command packet consists of eight values: the command byte, six parameter bytes, and the GO byte. The parameter bytes were discussed in detail in Chapter IV, Section B-4. The GO byte can take on any value and Ø was used. The command byte used in this implementation can be either 47H for the write operation or 4EH for the read operation. These values give both the read and write operations the desired characteristics of single sector processing, programmed I/O data transfers, and automatic retries enabled.

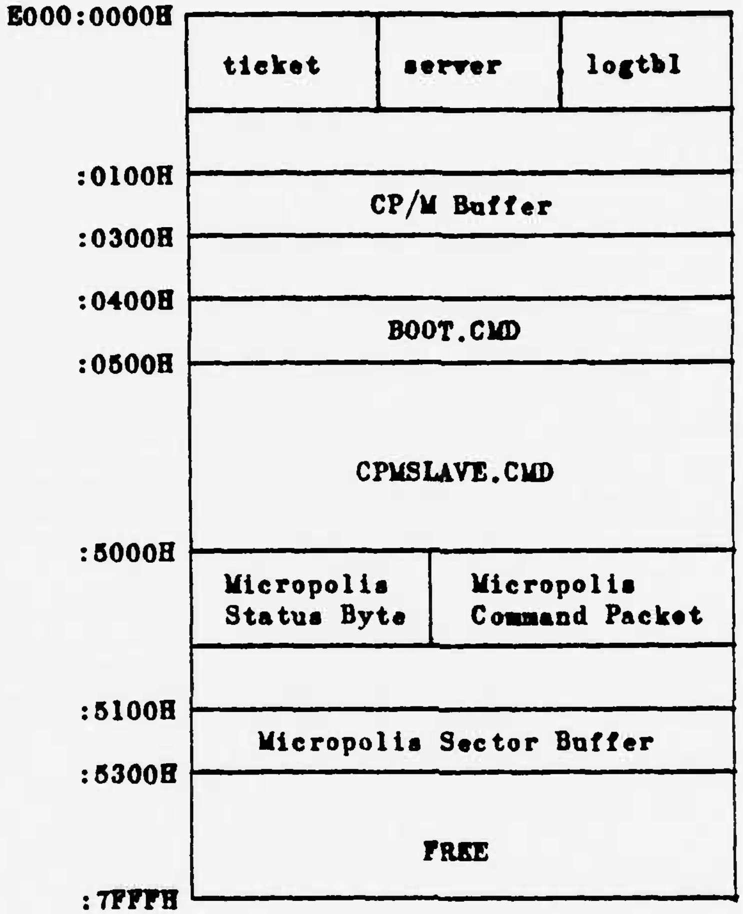


Figure 5.1 Final Common Memory Allocation Map

The sector buffer is used to transfer data to and from the controller. Both primary and alternate data transfer protocols are possible in the programmed I/O mode and these are shown in Figures 5.2 and 5.3 respectively. The alternate protocol differs from the primary protocol in the amount of status checking required. As shown, the primary protocol requires a status check between the transfer of each data byte while the alternate does not. Use of the alternate protocol is possible only if the loop time is greater than the 1.5 microseconds/byte response time of the controller. Recall from Chapter IV that all operations require a minimum of a MOV and an OUT instruction and these two instructions need 4.0 microseconds to execute. Thus, the alternate data protocol was employed in the interrupt handler to improve response times.

A brief description of how the resulting interrupt handler works is in order. When the timer-controlled interrupt occurs, the interrupt handler routine will save those registers that are needed by the routine on the stack and check the common memory status byte. If a non-zero value is found, the timer count value is re-loaded, the registers are restored from the stack, and a return is executed with no further action taken. If a zero value is found, the command byte is read to determine the direction of data transfer (to the common memory sector buffer for a read operation and from it for a write operation). Next,

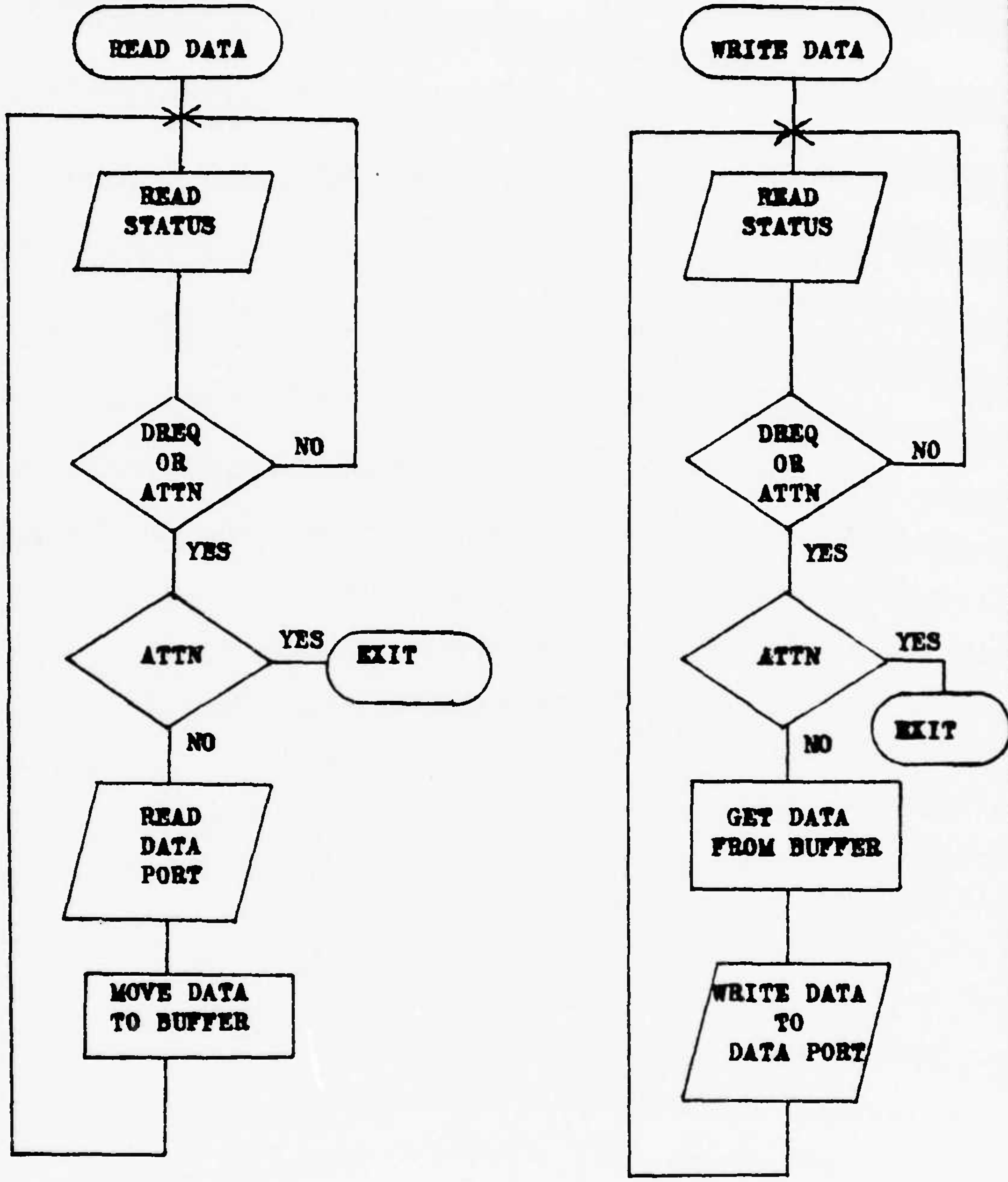


Figure 5.2 Primary Data Transfer Protocol

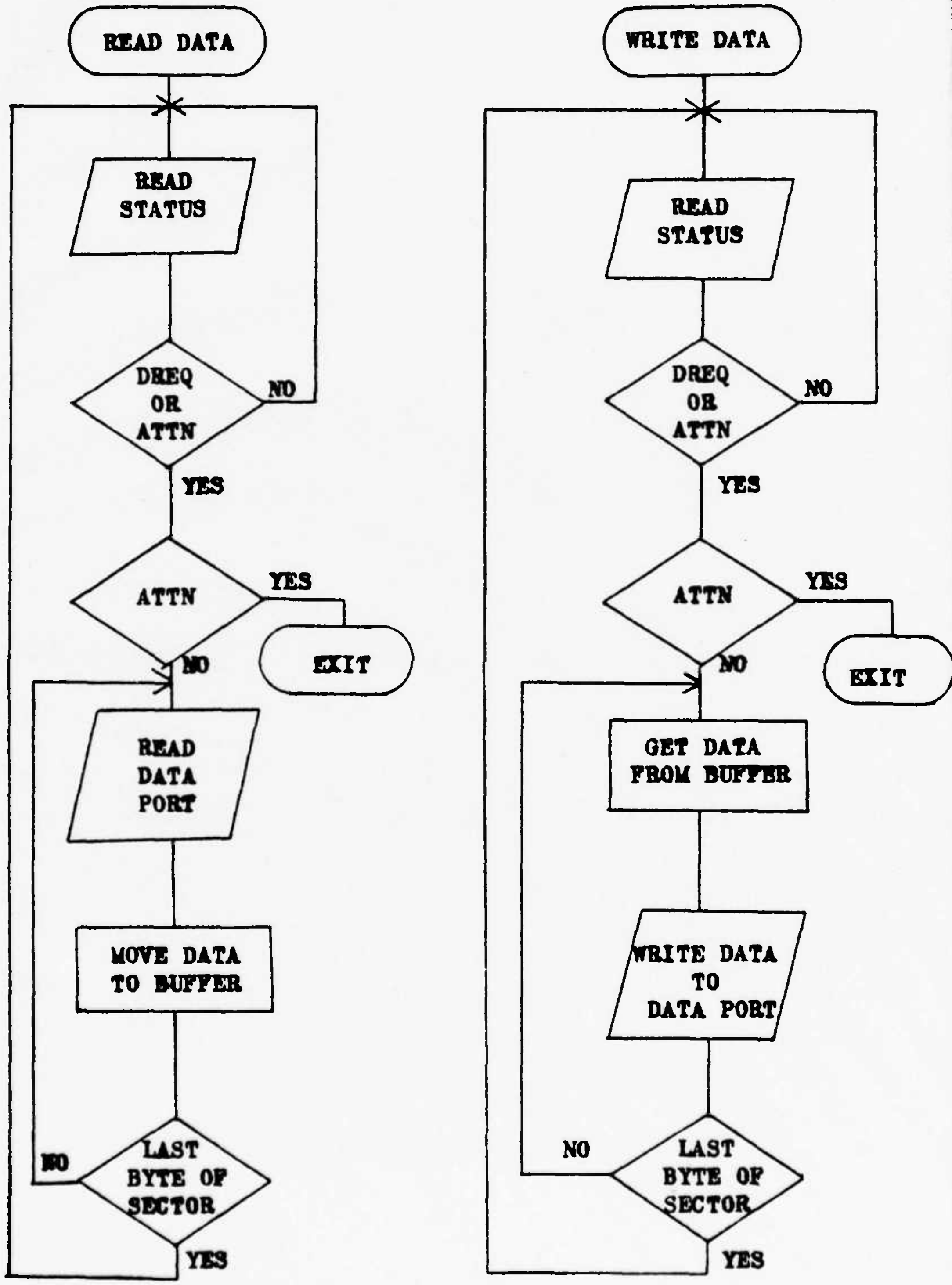


Figure 5.3 Alternate Data Transfer Protocol

the entire command packet is transmitted to the controller and when the DREQ signal is sensed, the data is transferred using the alternate protocol described above. Finally, the Termination Status byte is read from the controller and placed in the common memory status byte, the registers are restored from the stack, the timer count value is re-loaded, and a return is executed.

The interrupt handler and initialization routines were loaded into memory and DDT86.CMD was used to manually set the interrupt handler's common memory variables. This enabled extensive testing to be carried out and the routine was found to function as designed.

At this point, the interrupt handler was being loaded into local RAM at the top of the transient program area (TPA). Because all users have access to this area, it would be quite a simple matter to write over or otherwise alter the routine and thus, disable the disk communications capability. A more practical solution had to be found. It was decided to place the interr pt handler immediately following the return instruction of the Micropolis INIT routine for two reasons. First, because the routine would be a part of the operating system, it would not be as easily accessible by the user. Secondly, this co-locates the routine with the initialization of the hardware used to invoke it. This makes future modifications to the routine or the technique easier.

2. Blocking/Deblocking

The physical to logical sector mapping required for the Micropolis disk system was supplied, in part, by the Digital Research DEBLOCK.LIB source file. This file is supplied as a part of the CP/M-86 operating system. It provides a complete routine for HOME, SETTRK, SETSEC, and SELDSK operations. READ and WRITE operations are also supplied but require the user developed routines READHST and WRITEHST.

The READHST routine that was developed prepares the command packet described in Section 1 above and sends it to the common memory. The status byte is then changed to 0 and the routine waits for the interrupt handler to change the status byte. If the status byte indicates success, the common memory buffer is copied into a local sector buffer and the operation is complete. If an error is indicated, the error flag used by the BDOS is set and the common memory buffer is not copied.

The WRITEHST routine closely follows the READHST routine. First, a command packet is prepared and sent to the common memory. Then the local sector buffer is placed in the common memory sector buffer and the status byte is set to 2. The routine then waits for the status byte to indicate success or failure and the BDOS error flag is set accordingly. This completes the write operation.

It should be noted that in the READHST and WRITEHST routines, the status byte must be set to Ø only after the command packet and common memory sector buffer (write operations only) are set up. This is necessary because when the interrupt occurs, the interrupt handler will use the command packet and sector buffer in common memory at that time. If the status byte is set to Ø prior to preparing common memory, there may or may not be enough time to prepare it. Setting the status byte to Ø after the common memory has been prepared ensures that the intended operation will be carried out.

The DEBLOCK.LIB routines cited above were used in their entirety to provide the remaining device dependent routines. However, one minor alteration was necessary. All variable names were prefixed with "MIC". This was required because Almquist and Stevens' work used the same routines for the REMEX hard disk [Ref. 5].

C. INTEGRATION INTO THE MULTI-USER SYSTEM

To integrate the Micropolis hard disk into the multi-user system, the seven device dependent routines were placed in a single file, MICHARD.A86. Next, the Disk Parameter Table was updated to reflect the Micropolis disk unit. Each one of the five drive heads was configured to be to a logical CP/M drive. The final logical to physical device mapping for the multiuser system is shown in Table 5.1. The

Logical Device Number	Logical Device Letter	Physical Device
0	A:	MBB-80 Bubble Memory
1	B:	REMEX Floppy Disk Drive
2	C:	REMEX Floppy Disk Drive
3	D:	REMEX Hard Disk Head 0
4	E:	REMEX Hard Disk Head 1
5	F:	REMEX Hard Disk Head 2
6	G:	REMEX Hard Disk Head 3
7	H:	Micropolis Hard Disk Head 0
8	I:	Micropolis Hard Disk Head 1
9	J:	Micropolis Hard Disk Head 2
10	K:	Micropolis Hard Disk Head 3
11	L:	Micropolis Hard Disk Head 4

Table 5.1 Logical to Physical Device Map

calls to REQUEST and RELEASE were placed in the READ and WRITE routines, the label table file, CPMMAST.CFG, was updated, and an 'INCLUDE MICHARD.A86' statement was placed in the BIOS module. The steps of Figure 3.4 were followed and a new CPM.SYS was generated. Note that in the original system of Almquist and Stevens, this file was titled "CPMMAST.CMD". It was necessary to change the name as a result of other work accomplished during this research effort. This is discussed in the next section.

The master conditional assembly switch in the BIOS module was set to false and a CPMSLAVE.SYS file was created. The slave version is different from the master version in that the Micropolis interrupt handler and hardware initialization, as well as the intialization of common memory synchronization variables, are not included.

When the system was initially tested, it loaded properly and only certain commands, such as DIR and REN, could be executed. Commands such as PIP and STAT would not. In fact, the system would come to a halt and required re-booting when these commands were attempted. Many experiments were conducted in an effort to find the source of this problem. It was discovered that only the built-in commands (DIR, REN, ERA, and TYPE) would execute.

The difference between a built-in command and a transient utility program is that the built-in command resides in memory as a part of the operating system while

the transient utility program resides on disk. Thus, transient utility programs, like PIP, must first be loaded in memory. This program then becomes the applications program of Figure 3.1. Any disk operations required in the process of executing the program must be performed by the BDOS module. The applications program indicates that a disk operation is required by first setting the CX register of the 8086 to the appropriate function number and then executing the software interrupt number 224. When interrupted, the BDOS module will carry out the indicated function.

This conflicts with hardware interrupts. Recall from Chapter II, Section A-4 that a software interrupt is of higher priority than a hardware interrupt. As shown in Figure 2.4, when a software interrupt is being processed, the hardware interrupts are not sensed. A transient utility program enters the BDOS by executing a software interrupt as described above to perform any read/write operation. In this implementation, the read/write operation for the Micropolis can be completed only if the hardware interrupt occurs. Therefore, a deadlock results.

This problem was solved by executing an INT 70 instruction on the master board whenever a Micropolis read/write operation is needed. This forces the interrupt handler to execute even though the entry to the BDOS has prevented the hardware from causing its execution. The

CPM.SYS and CPMSLAVE.CMD files were re-generated as stated above and tested again. All command file executions on the master board were successful. All four AEGIS system boards were then booted and the multiuser system was tested. Simultaneous operations conducted on these four boards were also successful and it was concluded that the Micropolis disk unit had been successfully integrated into the multiuser AEGIS development system.

D. A NEW BOOT ROM AND LOADER

Early in the development stages of the Micropolis interface, it was discovered that a power failure would destroy the contents of bubble memory. Since, the operating system was booted from this device, work could not be continued until the bubble memory had been re-formatted and re-loaded with the operating system. This is an extremely time consuming task and the method is detailed in [Ref. 5, Chapter IV, Section D]. Thus, it was considered desirable to be able to boot load the operating system from the REMEX floppy disk drive. A new boot ROM and loader routine were developed for that purpose.

1. Boot Loader

As stated in Chapter III, the 8K byte EPROM chips on the iSBC 86/12A contain the INTEL 957 monitor and control is passed to the monitor whenever the system RESET button is pushed. Both the bubble boot loader and the INTEL MDS

system boot loader co-exist with the monitor in this 8K EPROM space.

It was originally thought that the REMEX boot loader could simply be added to the EPROM. However, this was not possible because the space occupied by the monitor severely limits the space available for programs. The 8K EPROM chips occupy the address space between 0FE000H and 0FFFFFH. The 957 monitor occupies the space between 0FE000H and 0FFD22H and contains a set of jump vectors at the base of this address space. This leaves approximately 720 decimal bytes of space for boot loader programs. It was decided at this point to maintain the monitor and INTEL MDS system boot loader in the EPROM and to replace the bubble boot loader with the one for REMEX. This would preserve the flexibility of being able to boot load the 86/12A from two different systems instead of just one.

A boot loader simply loads the program located on the reserved system tracks of the disk into memory and passes control to it. It is the responsibility of this program, the loader, to load the actual operating system into memory and pass control to it. To develop a boot loader for the REMEX would require that a working system loader be placed on the system tracks of the diskette. Since one had not yet been written for the REMEX, the bubble system loader, LDRMB80.CMD, was used for development purposes. This would permit the boot loader under

development to load the bubble system loader from the REMEX floppy disk and this loader would then load the operating system from bubble memory. A boot loader program, RMXROM.A86, was successfully written, debugged, and tested using this technique. The source code for the INTEL MDS boot loader was then successfully integrated into RMXROM.A86.

2. System Loader

The system tracks of a single density, eight inch floppy disk have 6.5K bytes of storage capacity and the system loader must be limited to this size. To generate a loader BIOS module for the REMEX, a conditional assembly switch was added to the present BIOS module. The switch, "loader_bios", when set to true, only includes in the assembly the device dependent code related to the REMEX floppy disk drive. A new label table file and a new Disk Parameter Table were created. These files, LDRMAST.CFG and LDRMAST.LIB respectively, reflect the REMEX floppy disk as the only disk drive in the system. The steps of Figure 3.5 were used to create the loader (It should be noted here that the LDCPM.CMD provided by Digital Research for use in the loader, expects to find the system file as CPM.SYS and this was the reason for the name change cited in Section C above). The resulting loader, RMXLDR.CMD, was approximately 5K bytes and easily fit on the system tracks. The REMEX

system loader was debugged and tested using the REMEX boot loader as the test vehicle.

3. <u>Programming the EPROM</u>

With the boot loader and system loader routines complete, the EPROM chips could now be programmed. The 8K bytes of EPROM consist of four 2K byte INTEL 2716's. Because of the even-odd addressing used on the 86/12A, two of these occupy the even 4K byte address space while the other two occupy the odd 4K byte address space. Because the only space available for boot loaders is located entirely within the upper 4K bytes of EPROM, it was necessary only to modify the two 2716's occupying this address space.

DDT86.CMD was used to read the contents of the upper 4K bytes of the 86/12A's EPROM and this was saved as a CMD file. Again using DDT86.CMD, the existing boot loaders were removed from this file and the contents of RMXROM.CMD were inserted. This resulted in a single contiguous file containing the upper portion of the 957 monitor, a boot loader for the REMEX, a boot loader for the INTEL MDS system, and the jump vectors required by the 957 monitor.

Two routines were then written that split this file into two files: one containing the odd address bytes and the other containing the even address bytes. Two new INTEL 2716s were then programmed with the contents of these two files. The newly programmed chips were placed on the iSBC 86/12A and tested. The INTEL MDS system was successfully

boot loaded by typing the command "GFFD4:0" from the monitor and the AEGIS development system was successfully booted from the REMEX floppy disk drive by typing the command "GFFD4:4" from the monitor.

VI. RESULTS AND CONCLUSIONS

A. EVALUATION

Two tests were conducted to evaluate the performance of the AEGIS development system. The first test consisted of assembling a 6K byte file and recording the assembly time. This was done for both the Micropolis disk system and the REMEX Data Warehouse with one, two, three, and finally, four computers operating simultaneously. Timing was accomplished with a standard stopwatch. The results of this test are shown in Table 6.1

The second test consisted of file transfers using the PIP.CMD file utility. This represents a worst case situation as file transfers are I/O intensive. Four transfer directions were tested: REMEX to REMEX, Micropolis to REMEX, REMEX to Micropolis, and Micropolis to Micropolis. A single 27K byte file was used as the transfer file. Once again, the test was conducted with first one computer operating and then with two, three, and four computers operating simultaneously. The results of this test are also shown in Table 6.1.

As indicated by the assembly data, there is not a linear relationship between the number of computers in the system and the times required for assembly. In fact, with four computers, the time required for assembly of the 6K byte

Command	Execution times (in seconds)			
	One Computer Operating	Two Computers Operating	Three Computers Operating	Four Computers Operating
ASM86 REMEX	25.2	37.0	43.7	58.2
ASM86 Micropolis	30.2	49.2	62.4	76.5
PIP REMEX to REMEX	5.5	22.3	29.7	32.9
PIP Micropolis to REMEX	7.4	22.4	36.0	43.0
PIP REMEX to Micropolis	9.3	36.0	46.0	62.5
PIP Micropolis to Micropolis	11.5	38.9	48.6	67.3

Table 6.1 Test Data

file is roughly 2.5 times the time required with just one computer. This is accounted for by realizing that, except for the contention for shared resources (common memory and hence, disk access time), each computer can assemble the file independently of the others.

From the file transfer data, as well as the assembly data, it is immediately apparent that the REMEX Data Warehouse operates faster than the Micropolis disk system. However, this is not an order of magnitude as one might expect when comparing a DMA interfaced hard disk to one that is interfaced using programmed I/O. If the ten millisecond polled interrupt (used to communicate with the Micropolis) is taken into account, the Micropolis performance would come even closer to that of the REMEX. The reason that the programmed I/O interface performance is close to that of a DMA interface is that more time is expended in disk head movement than memory access.

B. GENERAL CONCLUSIONS

The primary goal of this thesis was met. A hardware interface was designed for the Micropolis disk drive using programmed I/O techniques and this was successfully integrated into the AEGIS multiuser system. The system was demonstrated with four independent users operating simultaneously. The addition of the Micropolis disk system to the AEGIS system provides an additional 35.6M bytes of

on-line storage capacity and should prove to be more than adequate for program and data storage. This frees the REMEX hard disk to be used entirely for the SPY-1A radar emulation rather than as a software storage media.

A boot ROM was also developed that allowed loading the operating system from the REMEX floppy disk drive. This proved to be more reliable than the bubble boot procedure used at the onset of this research. The bubble memory frequently "crashed" and required reformatting and reloading before it could be used again. The cause for this was never discovered except for noting that every time a power failure occurred, the bubble memory would crash. The board has on-board power failure protection circuitry. However, the facilities required to thoroughly test this circuitry were not available.

Future research involving the AEGIS multiuser system should include expansion of the current 8K EPROM to 16K and development of a boot loader that would allow booting from any of the devices in the system. As the current system stands, if the REMEX floppy disk drive fails, either a new boot ROM will have to be generated or the old bubble boot ROM will have to be restored. This may eventually prove to be too inflexible.

Additionally, some type of protection scheme needs to be implemented for common memory. Currently, there is no protection and a user program that has gone out of control

could quite easily destroy the data in common memory. This development would require that some type of hardware access control be designed and the BIOS module be modified to activate that hardware whenever common memory access is required.

APPENDIX A

USER'S MANUAL FOR THE AEGIS SYSTEM

A. SYSTEM CONFIGURATION

The AEGIS development system consists of: one bubble memory board, four INTEL iSBC 86/12A boards, the REMEX Data Warehouse, the Micropolis disk system, and a 32K byte common memory board. These boards must meet certain requirements in order to work properly in the system and these are described in the paragraphs that follow.

1. Master iSBC 86/12A

This board is used as the Micropolis disk unit interface and provides the bus clock and constant clock signals to the MULTIBUS. This board must be positioned in the iCS-80 chassis in an odd-numbered slot (the slots are numbered from 1 to 16 left to right). The board requirements are:

Remove jumpers:

E13-E14	E21-E25
E15-E16	E28-E29
E17-E18	E30-E31
E19-E20	E32-E33

Add jumpers:

E28-E15	E30-E17
E30-E25	

Add 7438 drivers in sockets:

XA11	XA12

This will set up the 8255 interface for the Micropolis disk. To provide the constant clock and bus clock:

Add jumpers:

E103-E104 E105-E106

This board must also contain the EPROM chips with the REMEX boot routines. The final requirement is that the local RAM be made inaccessible to the MULTIBUS. This is done by adding jumper E112-E114 and removing jumper E115-E128.

2. All Other iSBC 86/12A's

The remaining boards must have local RAM inaccessable to the MULTIBUS and must not provide any clock signals. To make the RAM inaccessible, add jumper E112-E114 and remove jumper E115-E128. To disable the clock signals, remove jumper E103-E104 and E105-E106.

3. REMEX Disk Drive

The Remex controller board mus be plugged into an odd-numbered slot in the iCS-80 chassis.

4. Bubble Memory

Bubble memory must be plugged into the slot containing the RUN/HALT switch (currently position 3).

5. 32K Byte RAM Board

This board can be plugged into any slot in the chassis and must be configured to start at address E0000H.

B. ACTIVATING THE SYSTEM

Before turning any power on, ensure that the RUN/HALT switch located on the front panel of the iCS-80 chassis is in the HALT position. Next power-on the equipment in the following order:

1. Apply power to the iCS-80 chassis by turning the OFF/ON/LOCK key to the ON position.

2. Turn on the REMEX disk by toggling the OFF/ON switch on the upper right of the front panel to ON.

3. Activate the Micropolis disk by toggling the switch on the right of its front panel to the up position.

4. Spin up the REMEX disk by placing the STOP/START switch located on the upper left panel to START. The green light over this switch will go out.

5. Turn on all CRT's.

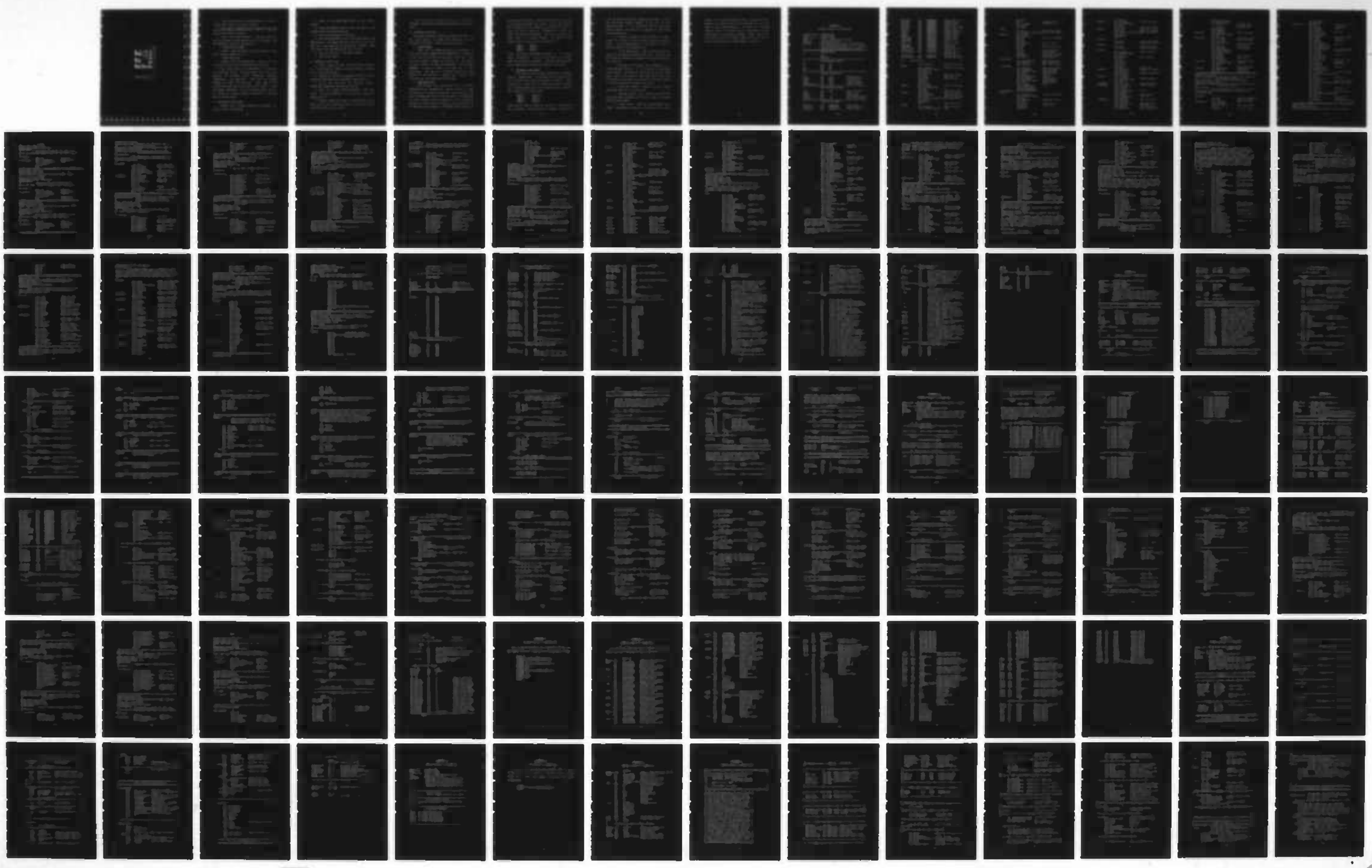
6. Toggle the RUN/HALT switch on the iCS-80 front panel to RUN.

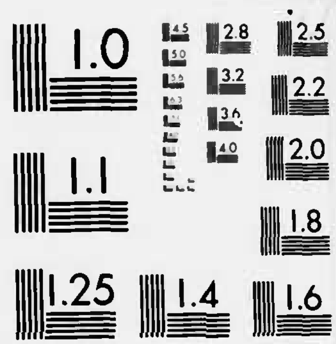
7. Press the master RESET switch on the iCS-80 panel in. This generally requires more than one RESET (normally three or four). The indicator of a good RESET is that all CRT's are printing stars and that the green lights over both the START/STOP switch and the A WRITE PROTECT switch of the REMEX are on.

With the power applied, the next step is to load the CP/M-86 operating system:

AD-A132 165 LOGIC DESIGN OF A SHARED DISK SYSTEM IN A MULTI-MICRO COMPUTER ENVIRONMENT(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA M L PERRY JUN 83 2/3

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

1. Place a system disk in the REMEX drive B (leftmost floppy drive) label up and close the door.

2. From the CRT connected to the master board (the one with the Miccropolis interface cable), type "U". This will activate the INTEL 957 monitor.

3. Enter the command "GFFD4:4". The disk in drive B will be accessed and approximately one to two minutes later the operating system will respond with:

ENTER CONSOLE NUMBER

4. Respond with the number on the front panel of the CRT. The next request will be:

ENTER LOGIN DISK

5. Respond with the desired disk. This will be the only disk that you will be permitted write access to.

6. The master board is now operational. To activate the remaining boards, first locate any disk drive that contains the following files: LDBOOT.CMD, BOOT.CMD, CPMSLAVE.SYS, and LDCPM.CMD. Then type the commands "LDBOOT" and "LDCPM" from that drive. Next, type "U" from any uninitialized board to activate its 957 monitor. Enter the command "GE000:400". As with the master board, you will be queried for a console number and login disk. Reply as with the master board.

C. DE-ACTIVATING THE SYSTEM

1. Ensure that no floppy disks are in the REMEX and that all users have finished.

2. Press in the master RESET button on the iCS-80 chassis.

3. Place the RUN/HALT switch on the front panel of the iCS-80 chassis to the HALT position.

4. Turn off the Micropolis disk unit.

5. Place the STOP/START switch of the REMEX disk in the STOP position. The green light over the switch will go out. When the light comes back on the disk has stopped. When this occurs, turn the REMEX power switch to OFF.

6. Turn the power off to the iCS-80 chassis.

7. Turn off all CRT's.

D. CREATING A SYSTEM DISK

1. First, format the disk. This will have to be done on a CP/M compatible system as the AEGIS system currently has no formatting routine.

2. Activate the system as described in Section B above. Place the formatted disk in the REMEX floppy drive B.

3. Locate any drive with the following files: LDCOPY.CMD, RMXLDR.CMD, CPM.SYS, and PIP.CMD.

4. Issue the command "LDCOPY RMXLDR.CMD" from this drive. You will be queried as to which drive to write to. Respond with "B".

5. Finally, issue the command "PIP B:=F:CPM.SYS" (Note here that "F" was an arbitrary choice, as any drive with the specified files on it will work).

6. You now have a system disk. It can be tested following the activation procedures described in Section B above.

E. MAINTENANCE UTILITIES

The system currently contains maintenance utilities for the bubble memory, the REMEX Data Warehouse, and the Micropolis disk system. These are described below.

1. <u>Bubble Memory</u>

There are two system utilities for maintenance of the bubble memory: DIAG86M.CMD and MBB8ØFMT.CMD. DIAG86M is a self test of the bubble and requires no user input other than the command "DIAG86" to activate it. Any faults occurring during this check are reported to the console. MBB8ØFMT is used to format the bubble for a CP/M environment. The user will be asked to enter the base address of the controller. Respond with "8ØØØH". The formatter will then format the bubble memory.

2. <u>REMEX Disk Drive</u>

Two routines are also available for the REMEX: RMXMAINT.CMD and RMXFORMT.CMD. Before either of these routines can be successfully executed, the local RAM of the board executing them must be made available to the MULTIBUS as these routines were not written to pass information through the common memory. Therefore, jumper E112-E114 must be removed prior to execution. RMXMAINT is menu driven with

nine available functions. Select the function from the list at the console and enter that number. Since these routines are carried out by the firmware of the REMEX controller no other user input is required. RMXFORMT will format the REMEX for the CP/M environment. You will be queried as to which "head" of the disk to format. The head to CP/M logical device is given in the following list:

Head 0	Drive D
Head 1	Drive E
Head 2	Drive F
Head 3	Drive G

Select the desired head number and enter it. No further inputs are required. You will be notified at the console when the formatting is complete. Restore jumper E112-E114 after completing all desired maintenance on the REMEX.

3. <u>Micropolis Disk System</u>

The Micropolis has a single menu-driven maintenance program, MICMAINT.CMD. This program must be executed only from the board containing the Micropolis interface. Prior to running it, ensure that all other system users are logged into a non-Micropolis disk. The Micropolis "head" number to CP/M logical disk is given below:

Head 0	Drive H
Head 1	Drive I
Head 2	Drive J
Head 3	Drive K
Head 4	Drive L

There are two types of commands in the menu: initialization and formatting. The initialization commands prepare the

disk for use and verify that there are no surface defects. The formatting commands prepare the disk for the CP/M operating system. If it is desired to run a initialization type of command, six values will be requested. These values are described below.

a. Physical Address of Logical Sector 0: This allows for a variety of logical sector mappings on the disk itself. In this system however, this value is currently 0 and this should be the response used.

b. Sector Skew Factor: This enables the sector address to be physically skewed on the disk. Currently, the CP/M operating imposes its own skew factor and this value is also set to 0.

c. Location of the Spare: The Micropolis has a sector sparing capability. If a bad sector is found during initialization, the spare can be written in the bad sectors place. Until a bad sector is noted, this value is 24. This will write the spare sector at the end of the track (sectors 0 - 23 are the only ones accessible by CP/M).

d. Disk Head Number: Respond with the desired head number from the list given above.

e. Starting Track: The Micropolis has 580 total tracks per head, numbered from 0 to 579. Enter the desired starting track number.

f. Ending Track: Enter the desired ending track number. The selected command will be executed on all tracks

between the starting number and ending number.The format commands only require the last four entries from the list above with the same conditions. All format or initialize commands should be followed with the corresponding verify format/verify initialization command. These require the same entries as for the original command and ensure that the disk function selected has been properly carried out.

APPENDIX B

PROGRAM LISTING OF MICMAINT.A86

```
;Program Name    : MICMAINT.A86
;Date            : 9 April 1983
;Written by      : Mark L. Perry
;For             : Thesis (AEGIS Modeling Group)
;Advisor         : Professor Cotton
;Purpose         : This routine enables the initialization
;                : and formatting functions to be carried out
;                : for the Micropolis Disk.  It is completely
;                : menu-driven and explanatory in nature.
;***************************************************************
;                  EQUATES TABLE
;
                   cseg
                   org 100h
;---------------------------------------------------------------
;                  MISCELLANEOUS EQUATES
;
cr                 equ     0dh
lf                 equ     0ah
wip                equ     1ah
be                 equ     7
;---------------------------------------------------------------
;                  EQUATES FOR 8255 PIO
;
porte              equ     0ceh              ;command port
porta              equ     0c8h              ;bi-directional
portb              equ     0cah              ;output port
portc              equ     0cch              ;control/status
mode_2_0_out       equ     0c0h              ;mode for 8255
;---------------------------------------------------------------
;                  BDOS FUNCTION EQUATES
;
bdos               equ     224               ;bdos interrupt
bdos_0             equ       0               ;ret to ccp
bdos_1             equ       1               ;char input
bdos_9             equ       9               ;bdos string output
bdos_10            equ      10               ;bdos buffer input
;---------------------------------------------------------------
;                  MICROPOLIS EQUATES
;
rstrb_on           equ     00001010b         ;read signal
rstrb_off          equ     00000010b         ;read signal off
wstrb_on           equ     00000110b         ;write signal
```

```
wstrb_off       equ     00000010b       ;write signal off
mic_stat        equ     00000000b       ;status signal
mic_cmd         equ     00000000b       ;command signal
mic_data        equ     00000001b       ;data signal
strb_on         equ     00000010b       ;input latch signal
strb_off        equ     00000011b       ;latch signal off
ack_on          equ     00000100b       ;output signal
ack_off         equ     00000101b       ;output signal off
en_sel          equ     00000010b       ;select enable
stndrd          equ     00010110b       ;normal reset
irdy_mask       equ     00000001b       ;input ready
ordy_mask       equ     00000010b       ;output ready
busy_mask       equ     00010000b       ;busy
mask            equ     10100000b       ;attn or dreq
attn_mask       equ     10000000b       ;attn only
dreq_mask       equ     00100000b       ;dreq only
initial_cmd     equ     11010001b       ;initialize cmd
verify_cmd      equ     11010101b       ;verify cmd
init_ver_cmd    equ     11011001b       ;initialize and
                                        ;verify cmd
format_cmd      equ     01000111b       ;formatting cmd
ver_form_cmd    equ     01000011b       ;verify the format
;
;       Main Program
;
                call mic_init           ;initialize disk
main:
                mov cl,bdos_9           ;output first menu
                mov dx,offset menu_1
                int bdos
                mov cl,bdos_1
                int bdos                ;get user option
                mov ah,0h               ;clear ah
                cmp al,'0'              ;valid entry?
                jb main_1
                cmp al,'6'
                jbe main_2
main_1:         mov cl,bdos_9           ;output error msg
                mov dx,offset err_1
                int bdos
                jmp main                ;and start over
main_2:         sub al,30h              ;adjust to binary
                mov cmd_type,al         ;store command
                add al,al               ;adj for tbl entry
                mov bx,offset jmp_tabl_1;get jump vector
                add bx,ax
                mov cx,[bx]
                jmp cx                  ;and jump to loc
s_end:
                mov cl,bdos_9           ;end of session
                mov dx,offset end_msg   ;msg
```

```
                int bdos
                sti                         ;re-enable int
                mov cl,bdos_0               ;return to ccp
                mov dl,00h
                int bdos
descr:
                mov al,0                    ;initialize count
                mov bx,offset jmp_tabl_2;output description
descr_1:        mov dx,[bx]
                mov cl,bdos_9
                push ax                     ;save the registers
                push bx
                int bdos
                mov cl,bdos_1               ;wait on user
                int bdos
                pop bx                      ;restore the regs
                pop ax
                inc al                      ;test for end of
                cmp al,7                    ;description
                je main                     ;if end start over
                add bx,02                   ;else get next msg
                jmp descr_1                 ;and output it
in_ver_dsk:
                call log_sec0_num           ;get logical sec 0
                call skw_num                ;get skew factor
                call spar_loc               ;get loc of spare
fm_ver_dsk:
                call head_num               ;get disk head num
                mov cl,bdos_9               ;output prompt
                mov dx,offset msg_5         ;for beginning
                int bdos                    ;track number
                call trk_num                ;get it
                mov beg_trk_num,dx          ;store it
                mov cl,bdos_9               ;output prompt
                mov dx,offset msg_6         ;for ending
                int bdos                    ;track number
                call trk_num                ;get it
                cmp dx,beg_trk_num
                jae fm_ver_dsk_1
                xchg dx,beg_trk_num
fm_ver_dsk_1:   mov end_trk_num,dx
main_3:
                mov cl,bdos_9               ;output second menu
                mov dx,offset menu_2
                int bdos
                mov cl,bdos_1               ;get user option
                int bdos
                mov ah,00h                  ;clear ah
                cmp al,'0'                  ;valid entry?
                jb main_4
                cmp al,'4'
```

```
            jbe main_5
main_4:     mov cl,bdos_9             ;output error msg
            mov dx,offset err_1
            int bdos
            jmp main_3                ;and start again
main_5:     sub al,30h                ;adjust to binary
            add al,al                 ;adj for tbl entry
            mov bx,offset jmp_tabl_3;get jump vector
            add bx,ax
            mov cx,[bx]
            jmp cx
rev_ent:
            call rev                  ;output the review
            jmp main_3                ;second menu again
chg_ent:
            call chg                  ;get the change
            jmp main_3                ;second menu again
e_cmmd:
            mov cl,bdos_9             ;output warning
            mov dx,offset warn
            int bdos
            mov cl,bdos_1             ;get response
            int bdos
            cmp al,'y'
            jz e_cmmd_1
            cmp al,'Y'
            jz e_cmmd_1
            jmp main                  ;start over
e_cmmd_1:   cmp cmd_type,2            ;check for command
            ja e_cmmd_2
            call mic_conv1            ;prepare parameters
            jmp e_cmmd_3
e_cmmd_2:   call mic_conv2
            mov cl,bdos_9             ;output message
            mov dx,offset msg_8
            int bdos
e_cmmd_2a:  call mic_send             ;send parameters
e_cmmd_2b:  call mic_status           ;attn or dreq?
            test al,mask
            jz e_cmmd_2b
            test al,attn_mask         ;attn?
            jnz e_cmmd_2c
            mov al,0e5h               ;must be dreq
            call mic_data_out         ;send E5h
            jmp e_cmmd_2b
e_cmmd_2c:  call mic_busy             ;wait on cntrl
            call mic_irdy
            call mic_data_in          ;get term byte
            cmp al,00h                ;success?
            jnz cmmd_err              ;no, then error
            inc beg_trk_num           ;any tracks left?
```

```
                mov dx,beg_trk_num
                cmp dx,end_trk_num
                ja e_cmmd_3b                ;finished here
                                            ;so start over
                mov parm2,dl                ;adjust parms
                mov parm3,dh
                jmp e_cmmd_2a
e_cmmd_3:       mov cl,bdos_9               ;output message
                mov dx,offset msg_10
                int bdos
e_cmmd_3a:      call mic_send               ;send first parms
                call mic_busy               ;wait for cntrl
                call mic_irdy
                call mic_data_in            ;get term status
                cmp al,0                    ;successful?
                jnz cmmd_err                ; now error
                inc beg_trk_num             ;more tracks?
                mov dx,beg_trk_num
                cmp dx,end_trk_num
                ja e_cmmd_3b
                mov parm2,dl                ;adjust parm2
                mov parm3,dh                ;and parm3
                jmp e_cmmd_3a
e_cmmd_3b:      mov cl,bdos_9               ;output success
                mov dx,offset msg_11        ;message
                int bdos
                jmp main                    ;and start over
cmmd_err:       mov err_code,al             ;save error
                call proc_err               ;process it
                jmp main                    ;start over
;*************************************************************
;Subroutine: proc_err
;Entry conditions: an error has occurred in the execution
;                  of a command on the disk
;Exit conditions: 'proc_err_tabl' has been updated
;Registers altered: none
;Subroutines called: save,restor,bin_dec,dec_asc,mic_busy,
;                    mic_irdy,mic_data_in
;Description:
;               This routine provides as console output
;the details of an error condition as issued by the
;disk controller.
;
proc_err:
                call save                   ;save all registers
                mov bl,0                    ;set up counter
proc_err_1:     call mic_busy               ;wait on cntrl
                call mic_irdy
                call mic_data_in            ;get aux status
                inc bl
                cmp bl,6                    ;is it 6th one?
```

```
                 je proc_err_2
                 jmp proc_err_1
proc_err_2:      mov dl,al                   ;put sector in dl
                 mov dh,00h                  ;clear dh
                 call bin_dec                ;convert it
                 call dec_asc
                 mov asc_sec,dl              ;store it
                 mov asc_sec_1,bh
                 mov asc_sec_2,bl
                 mov dl,head                 ;get head number
                 mov dh,00h                  ;clear dh
                 call bin_dec                ;convert it
                 call dec_asc
                 mov asc_dk_head,dl          ;store it
                 mov asc_dk_head_1,bh
                 mov asc_dk_head_2,bl
                 mov dx,beg_trk_num          ;get track number
                 call bin_dec                ;convert it
                 call dec_asc
                 mov asc_trk,dl              ;store it
                 mov asc_trk_1,bh
                 mov asc_trk_2,bl
                 mov dl,err_code             ;get error code
                 mov dh,00h                  ;clear dh
                 call bin_dec                ;convert it
                 call dec_asc
                 mov asc_err_c,dl            ;store it
                 mov asc_err_c_1,bh
                 mov asc_err_c_2,bl
                 mov cl,bdos_9               ;output cmd type
                 mov dx,offset procerrtabl
                 int bdos
                 mov ah,00h                  ;clear ah
                 mov al,cmd_type
                 add al,al                   ;adjust for table
                 mov bx,offset jmp_tabl_4;get jump vector
                 add bx,ax
                 mov dx,[bx]
                 mov cl,bdos_9               ;output it
                 int bdos
                 mov cl,bdos_9               ;rest of table
                 mov dx,offset procerrtabl1
                 int bdos
                 mov cl,bdos_1               ;wait on user
                 int bdos                    ;to read it
                 call restor                 ;restore registers
                 ret
;*****************************************************************
;Subroutine: mic_busy
;Entry conditions: none
;Exit conditions: disk controller has issued 'not busy'
```

```
;                    signal
;Registers altered: none
;Subroutines called: mic_status
;Description:
;                    The executing program will wait here
;until the disk controller issues the 'not busy' signal.
;
mic_busy:
                    push ax                     ;save ax
mic_busy_1:         call mic_status             ;get status
                    test al,busy_mask           ;busy?
                    jz mic_busy_1
                    pop ax
                    ret
;*************************************************************
;Subroutine:  mic_irdy
;Entry conditions: none
;Exit conditions: disk controller has issued 'irdy'
;                    signal
;Registers altered: none
;Subroutines called: mic_status
;Description:
;                    The execution of the program will
;wait here until 'irdy' is issued by the controller.
;
mic_irdy:
                    push ax                     ;save ax
mic_irdy_1:         call mic_status             ;get status
                    test al,irdy_mask           ;ready?
                    jz mic_irdy_1
                    pop ax                      ;restore ax
                    ret                         ;ready now
;*************************************************************
;Subroutine: mic_ordy
;Entry conditions: none
;Exit conditions: disk controller has issued the 'ordy'
;                    signal
;Registers altered: none
;Subroutines called: mic_status
;Description:
;                    The execution of the program will wait
;here until 'ordy' is issued by the controller.
;
mic_ordy:
                    push ax                     ;save ax
mic_ordy_1:         call mic_status             ;get status
                    test al,ordy_mask           ;ready?
                    jz mic_ordy_1               ;not yet
                    pop ax
                    ret
;*************************************************************
```

```
;Subroutine: mic_send
;Entry conditions: parameters are calculated and in
;                  the byte variables
;Exit conditions: parameters and command have been sent
;Registers altered: none
;Subroutines called: save,restor,mic_busy,mic_ordy,
;                    mic_irdy,mic_cmd_out,mic_data_out
;Description:
;                The command byte, six parameter bytes
;and the go byte found in the data area are sent to
;the disk controller.
;
mic_send:
                call save               ;save registers
                call mic_busy           ;wait for cntrl
                call mic_ordy
                call mic_cmd_out        ;send out cmd
                mov bx,offset parm1     ;send parameters
                mov dl,0                ;counter
mic_send_1:     call mic_busy           ;wait for cntrl
                call mic_ordy
                mov al,[bx]             ;get parm
                call mic_data_out       ;send it
                inc bx
                inc dl
                cmp dl,7                ;done?
                jb mic_send_1
                call restor             ;restore registers
                ret
;*************************************************************
;Subroutine: mic_cmd_out
;Entry conditions: 'ordy' signal has been issued by the
;                  disk controller and 'cmd_byte'
;                  contains the command to be sent.
;Exit conditions: none
;Registers altered: none
;Subroutines called: none
;Description:
;       The command in the byte variable 'cmd_byte'
;is sent to the disk controller.
;
mic_cmd_out:
                push ax                 ;save ax
                mov al,cmd_byte
                out porta,al            ;to bi-directional
                mov al,mic_cmd          ;enable cmd line
                out porte,al
                mov al,ack_on           ;activate output
                out porte,al            ;buffer
                mov al,wstrb_on         ;pulse the write
                out portb,al            ;strobe
```

```
                mov al,wstrb_off
                out portb,al
                mov al,ack_off              ;de-activate the
                out porte,al                ;output buffer
                pop ax
                ret
;*************************************************************
;Subroutine: mic_data_in
;Entry conditions: 'irdy' signal has been issued by the
;                  disk controller
;Exit conditions: al contains data byte
;Registers altered: al
;Subroutines called: none
;Description:
;               A byte of data is input from the Micropolis
;disk unit.
;
mic_data_in:
                mov al,mic_data             ;enable the data
                out porte,al
                mov al,rstrb_on             ;turn the read
                out portb,al                ;on
                mov al,strb_on              ;latch the data
                out porte,al
                mov al,strb_off
                out porte,al
                mov al,rstrb_off            ;turn off the
                out portb,al                ;read signal
                in al,porta                 ;bring in data
                ret
;*************************************************************
;Subroutine: mic_data_out
;Entry conditions: 'ordy' signal has been issued by the
;                  disk controller and al contains value
;                  to be sent.
;Exit conditions: none
;Registers altered: none
;Subroutines called: none
;Description:
;               A byte of data is output to the Micropolis
;disk unit.
;
mic_data_out:
                push ax                     ;save ax
                out porta,al                ;to bi-directional
                mov al,mic_data             ;enable data line
                out porte,al
                mov al,ack_on               ;activate output
                out porte,al                ;buffer
                mov al,wstrb_on             ;pulse the write
                out portb,al                ;strobe
```

```
                mov al,wstrb_off
                out portb,al
                mov al,ack_off               ;de-activate
                out porte,al                 ;output buffer
                pop ax                       ;restore value
                ret
;***********************************************************
;Subroutine: mic_conv1
;Entry conditions: none
;Exit conditions: parameters are set for disk use
;Registers altered: none
;Subroutines called: save,restor
;Description:
;               This subroutine prepares the parameters
;required by the Micropolis disk drive for verify or
;initialization commands.
;
mic_conv1:
                call save                    ;save the registers
                mov al,cmd_type              ;check for command
                cmp al,0                     ;initialize?
                jz mic_conv1_1
                cmp al,1                     ;verify?
                jz mic_conv1_2
                mov cmd_byte,init_ver_cmd ;initialize
                jmp mic_conv1_3               ;and verify
mic_conv1_1:    mov cmd_byte,initial_cmd;it was initialize
                jmp mic_conv1_3              ;only
mic_conv1_2:    mov cmd_byte,verify_cmd ;it was verify
mic_conv1_3:    mov al,head                  ;prepare head num
                mov cl,4
                sal al,cl
                mov parm1,al
                mov dx,beg_trk_num           ;set up parameter
                mov parm2,dl                 ;2 and 3
                mov parm3,dh
                mov al,log_sec0              ;set up parameter
                mov parm4,al                 ;4
                mov al,skw_fac               ;set up parameter
                mov parm5,al                 ;5
                mov al,spar                  ;set up parameter
                mov parm6,al                 ;6
                mov go_byte,0                ;set up go byte
                call restor                  ;restore registers
                ret
;***********************************************************
;Subroutine: mic_conv2
;Entry conditions: none
;Exit conditions: parameters are set for disk
;Registers altered: none
;Subroutines called: save,restor
```

```
;Description:
;               This subroutine prepares the parameters for
;the Micropolis disk drive for format and verify format
;commands.
;
mic_conv2:
                call save                       ;save the regs
                mov al,cmd_type
                cmp al,3                        ;format?
                jz mic_conv2_1
                mov cmd_byte,ver_form_cmd;must be verify
                jmp mic_conv2_2
mic_conv2_1:    mov cmd_byte,format_cmd
mic_conv2_2:    mov al,head                     ;set parameter 1
                mov cl,4                        ;adjust position
                sal al,cl
                mov parm1,al
                mov dx,beg_trk_num              ;set parameter 1
                mov parm2,dl                    ;and 2
                mov parm3,dh
                mov parm4,0                     ;starting sector
                mov parm5,24                    ;process 24
                mov parm6,0                     ;not used
                mov go_byte,0
                call restor                     ;restore registers
                ret
;**************************************************************
;Subroutine: mic_init
;Entry conditions: none
;Exit conditions: disk has been initialized
;Registers altered: ax,cx
;Subroutines called: mic_status
;Description:
;               This subroutine resets and initializes the
;Micropolis disk drive and the 8255 parallel i/o port.
;If the reset attempt fails, the program is aborted and
;the user is returned to the operating system.
;
mic_init:
                cli                             ;disable maskable
                                                ;interrupts
                mov al,mode_2_0_out             ;initialize to mode
                out portc,al                    ;0 and 2
                mov al,ack_off                  ;insure acknowledge
                out portc,al                    ;is off
                mov al,strb_off                 ;insure strobe
                out portc,al                    ;is off
                mov al,en_sel                   ;set select and
                out portb,al                    ;enable
                mov cx,10                       ;wait 1 second
mic_init_1:     mov ax,27777
```

```
mic_init_2:     dec ax
                jnz mic_init_2
                dec cx
                jnz mic_init_1
                call mic_status         ;get the status
                cmp al,stndrd
                jz mic_init_3           ;then return
                mov cl,bdos_9           ;output error
                mov dx,offset micrst_err;message
                int bdos
                mov cl,00               ;and return to
                mov dl,00               ;o/s
                int bdos
mic_init_3:     ret
;*************************************************************
;Subroutine: mic_status
;Entry Conditions: none
;Exit Conditions: al contains status of disk
;Registers altered: al
;Subroutines called: none
;Description:
;       This subroutine reads and returns the current
;value of the Micropolis disk controller's status port.
;
mic_status:
                mov al,mic_stat         ;enable stat line
                out portc,al
                mov al,rstrb_on         ;turn on read
                out portb,al
                mov al,strb_on          ;latch the status
                out portc,al
                mov al,strb_off
                out portc,al
                mov al,rstrb_off        ;turn off read
                out portb,al
                in al,porta             ;bring in status
                ret
;*************************************************************
;Subroutine: chg
;Entry conditions: none
;Exit conditions: desired value is changed to new value
;Registers altered: none
;Subroutines called: save,restor,log_sec0_num,skw_num,
;                    spar_loc,head_num,trk_num
;Description:
;               This subroutine allows the user to change
;a value that has been previously specified by a call to
;the appropriate routine.
;
chg:
                call save               ;save registers
```

```
                cmp cmd_type,2             ;two diff opts
                ja cng_4                   ;depending on cmd
cng_1:          mov cl,bdos_9              ;output menu
                mov dx,offset menu_3
                int bdos
                mov cl,bdos_1              ;get user option
                int bdos
                mov ah,0h                  ;clear ah
                cmp al,'0'                 ;valid entry?
                jb cng_2
                cmp al,'5'
                jbe cng_3
cng_2:          mov cl,bdos_9              ;output error
                mov dx,offset err_1
                int bdos
                jmp cng_1                  ;and start over
cng_3:          sub al,30h                 ;convert to binary
                add al,al                  ;adjust for table
                mov bx,offset jmp_tabl_5;get jump vector
                add bx,ax
                mov cx,[bx]
                jmp cx
cng_4:          mov cl,bdos_9              ;output menu
                mov dx,offset menu_4
                int bdos
                mov cl,bdos_1              ;get option
                int bdos
                mov ah,0h                  ;clear ah
                cmp al,'0'                 ;valid entry?
                jb cng_5
                cmp al,'2'
                jbe cng_6
cng_5:          mov cl,bdos_9              ;output error
                mov dx,offset err_1
                int bdos
                jmp cng_4
cng_6:          sub al,30h                 ;convert to binary
                add al,al                  ;adjust for table
                mov bx,offset jmp_tabl_6;get jump vector
                add bx,ax
                mov cx,[bx]
                jmp cx
cng_7:          call log_sec0_num          ;get new logical
                jmp cng_13                 ;sector number
cng_8:          call skw_num               ;get new skew
                jmp cng_13                 ;factor
cng_9:          call spar_loc              ;get new spare
                jmp cng_13                 ;location
cng_10:         call head_num              ;get new head
                jmp cng_13                 ;number
cng_11:         mov cl,bdos_9              ;get new beginning
```

```
                mov dx,offset msg_5      ;track number
                int bdos
                call trk_num
                cmp dx,end_trk_num
                jbe cng_11a
                xchg dx,end_trk_num
cng_11a:        mov beg_trk_num,dx
                jmp cng_13
cng_12:         mov cl,bdos_9            ;get new ending
                mov dx,offset msg_5      ;track number
                int bdos
                call trk_num
                cmp dx,beg_trk_num
                jae cng_12a
                xchg dx,beg_trk_num
cng_12a:        mov end_trk_num,dx
cng_13:         call restor              ;restore registers
                ret
;**********************************************************************
;Subroutine: rev
;Entry conditions: none
;Exit conditions: none
;Registers altered: none
;Subroutines called: save,restor,bin_dec,dec_asc
;Description:
;       This subroutine prints out at the console a
;complete tabulation of user supplied input and
;returns to the calling program.
;
rev:
                call save                ;save all regs
                mov dh,00h               ;convert logical
                mov dl,log_sec0          ;sector to asc
                call bin_dec
                call dec_asc
                mov asc_log_sec0,dl      ;store it
                mov asc_log_sec0_1,bh
                mov asc_log_sec0_2,bl
                mov dh,00h               ;convert skew
                mov dl,skw_fac           ;factor to asc
                call bin_dec
                call dec_asc
                mov asc_skw_fac,dl       ;store it
                mov asc_skw_fac_1,bh
                mov asc_skw_fac_2,bl
                mov dh,00h               ;convert location
                mov dl,spar              ;of spare to asc
                call bin_dec
                call dec_asc
                mov asc_spar,dl          ;store it
                mov asc_spar_1,bh
```

```
                mov asc_spar_2,bl
                mov dh,00h                  ;convert disk
                mov dl,head                 ;head to asc
                call bin_dec
                call dec_asc
                mov asc_head,dl             ;store it
                mov asc_head_1,bh
                mov asc_head_2,bl
                mov dx,beg_trk_num          ;convert beginning
                call bin_dec                ;trk to asc
                call dec_asc
                mov asc_beg_trk,dl          ;store it
                mov asc_beg_trk_1,bh
                mov asc_beg_trk_2,bl
                mov dx,end_trk_num          ;convert ending
                call bin_dec                ;trk to asc
                call dec_asc
                mov asc_end_trk,dl          ;store it
                mov asc_end_trk_1,bh
                mov asc_end_trk_2,bl
                mov cl,bdos_9               ;output command
                mov dx,offset rev_tabl      ;type
                int bdos
                mov ah,0                    ;clear ah
                mov al,cmd_type
                add al,al                   ;adjust for table
                mov bx,offset jmp_tabl_4;get jump vector
                add bx,ax
                mov dx,[bx]
                mov cl,bdos_9               ;output command
                int bdos                    ;name
                mov cl,bdos_9               ;output the
                cmp cmd_type,2              ;entire table now
                ja rev_1
                mov dx,offset rev_tabl_1
                int bdos
                jmp rev_2
rev_1:          mov dx,offset rev_tabl_2
                int bdos
rev_2:          mov cl,bdos_1               ;wait on user to
                int bdos                    ;read it
                call restor                 ;restore registers
                ret
;***********************************************************
;Subroutine: spar_loc
;Entry conditions: none
;Exit conditions: 'spar' contains value for location
;                 of spare sector
;Registers altered: none
;Subroutines called: save,restor,con_in,asc_dec,dec_bin
;Description:
```

```
;       The user is prompted for the location of the spare
;sector.  The valid range is 0 to 255.  A number outside
;of this range results in an error message and another
;prompt.  The valid number is converted to binary and
;stored in the byte variable 'spar'.
;
spar_loc:
                call save                       ;save all registers
spar_loc_1:     mov cl,bdos_9                   ;output prompt
                mov dx,offset msg_7
                int bdos
                call con_in                     ;get response
                call asc_dec                    ;convert it
                call dec_bin
                cmp dx,255                      ;ck for range
                jbe spar_loc_2
                mov cl,bdos_9                   ;output error msg
                mov dx,offset err_2
                int bdos
                jmp spar_loc_1                  ;start over
spar_loc_2:     mov spar,dl                     ;store in 'spar'
                call restor                     ;restore registers
                ret
;****************************************************************
;Subroutine: trk_num
;Entry conditions: none
;Exit conditions: dx contains a track number
;Registers altered: dx
;Subroutines called: save,restor,con_in,asc_dec,dec_bin
;Description;
;       The user is prompted for a track number.  The
;valid range is 0 to 579.  Invalid input results in an
;error message and another prompt.  The valid number
;is converted to binary and returned in dx.
;
trk_num:
                call save                       ;save all regs
trk_num_1:      mov cl,bdos_9                   ;output prompt
                mov dx, offset msg_4
                int bdos
                call con_in                     ;get response
                call asc_dec                    ;convert it
                call dec_bin
                cmp dx,579                      ;ck for range
                jbe trk_num_2
                mov cl,bdos_9                   ;output error msg
                mov dx,offset err_2
                int bdos
                jmp trk_num_1                   ;start over
trk_num_2:      mov temp_tk,dx                  ;save trk number
                call restor                     ;restore the regs
```

```
                mov dx,temp_tk           ;restore dx
                ret
;*****************************************************************
;Subroutine: skw_num
;Entry conditions: none
;Exit conditions: 'skw_fac' contains sector skew
;                 factor
;Registers altered: none
;Subroutines called: save,restor,con_in,asc_dec,dec_bin
;Description:
;       The user is prompted for a sector skew factor.
;The valid range is 0 to 23.  A number outside of this
;range results in an error message and another prompt.
;The valid number is converted to binary and stored in
;the byte variable 'skw_fac'.
;
skw_num:
                call save                ;save all regs
skw_num_1:      mov cl,bdos_9            ;output prompt
                mov dx,offset msg_3
                int bdos
                call con_in              ;get response
                call asc_dec             ;convert it
                call dec_bin
                cmp dx,23                ;ck for range
                jbe skw_num_2
                mov cl,bdos_9            ;output error msg
                mov dx,offset err_2
                int bdos
                jmp skw_num_1            ;start over
skw_num_2:      mov skw_fac,dl           ;store in 'skw_fac'
                call restor              ;restore the regs
                ret
;*****************************************************************
;Subroutine: log_sec0_num
;Entry conditions: none
;Exit conditions: 'log_sec0' contains address of logical
;                 sector 0
;Registers altered: none
;Subroutines called: save,restor,con_in,asc_dec,dec_bin
;Description:
;       The user is prompted to input the physical address
;of logical sector 0.  This number can be in the range 0 to
;23.  The input is checked and an error message results if
;it is invalid.  The user is also prompted again in this
;event.  The valid number is converted to binary and stored
;in the byte variable 'log_sec0'
;
log_sec0_num:
                call save                ;save all regs
log_sec0_num_1: mov cl,bdos_9            ;output prompt
```

```
                mov dx,offset msg_2
                int bdos
                call con_in                 ;get response
                call asc_dec                ;convert it
                call dec_bin
                cmp dx,23                   ;ck for range
                jbe log_sec0_num_2
                mov cl,bdos_9               ;output error msg
                mov dx,offset err_2
                int bdos
                jmp log_sec0_num_1
log_sec0_num_2: mov log_sec0,dl             ;st in 'log_sec0'
                call restor                 ;restore regs
                ret
;*************************************************************
;Subroutine: head_num
;Entry conditions: none
;Exit conditions: 'head' contains head number
;Registers Altered: none
;Subroutines called: con_in, asc_dec, dec_bin, save, restor
;Description:
;The user is prompted to input a head number in the range
;of 0 to 4.  The input is checked ,if an invalid number is
;entered, an error message is output and the user is again
;prompted for an entry.  The valid number is converted to
;binary and stored in the byte variable 'head'.
;
head_num:
                call save                   ;save all regs
head_num_1:     mov cl,bdos_9               ;output prompt
                mov dx,offset msg_1
                int bdos
                call con_in                 ;get response
                call asc_dec                ;convert to decimal
                call dec_bin                ;convert to binary
                cmp dx,0004h                ;check for range
                jbe head_num_2
                mov cl,bdos_9               ;output error msg
                mov dx,offset err_2
                int bdos
                jmp head_num_1              ;and start over
head_num_2:     mov head,dl                 ;store in 'head'
                call restor                 ;restore registers
                ret
;*************************************************************
;Subroutine: con_in
;Entry conditions: none
;Exit conditions: dx contains most significant ASCII
;                    digits entered
;                 bx contains least significant ASCII
;                    digits entered
```

```
;Registers altered: dx,bx
;Subroutines called: save,restor
;Deescription:
;               BDOS function 10 is utilized to input
;a line of edited data from the console.  Backspacing
;is permitted through the use of Control-H or Control-X.
;Only a maximum of 3 characters can be entered.  To
;alter this, the value of 'buffer' must be changed.  For
;a complete description of BDOS function 10, see page 29
;in "CP/M-86 Operating System System Guide" by Digital
;Research.  Two error conditions are reported: (1) if
;no data has been entered and (2) if the data entered
;is non-numerical.  In each case the user is prompted
;for data again.
;
con_in:
                call save               ;save all regs
con_in_1:       mov cl,bdos_10          ;bdos console in
                mov dx,offset buffer    ;input buffer
                mov buffer,3            ;max char count
                int bdos
                cmp num_chars,0         ;ck for no chars
                jne con_in_3
con_in_2:       mov cl,bdos_9           ;console output
                mov dx,offset err_in    ;loc of err msg
                int bdos
                jmp con_in_1
con_in_3:       mov dl,num_chars        ;check each char
                mov bx,offset asc_data_1;entered for
con_in_4:       mov al,[bx]             ;valid asc number
                cmp al,'0'
                jb con_in_2
                cmp al,'9'
                ja con_in_2
                inc bx                  ;get next number
                dec dl                  ;test for last num
                jz con_in_5
                jmp con_in_4
con_in_5:       mov dx,0                ;initialize result
                mov bx,0
                cmp num_chars,1         ;ck for 1 char
                jne con_in_6
                mov bl,asc_data_1
                jmp con_in_8
con_in_6:       cmp num_chars,2         ;ck for 2 chars
                jne con_in_7
                mov bh,asc_data_1
                mov bl,asc_data_2
                jmp con_in_8
con_in_7:       mov dl,asc_data_1       ;must be 3 chars
                mov bh,asc_data_2
```

```
                mov bl,asc_data_3
con_in_8:       mov ms_data,dx           ;save result
                mov ls_data,bx
                call restor
                mov dx,ms_data           ;place in dx
                mov bx,ls_data           ;and bx
                ret
;*************************************************************************
;Subroutine: asc_dec
;Entry conditions:dx contains ASCII representation of most
;                    significant 2 digits of 4 digit number
;                 bx contains ASCII representation of least
;                    significant 2 digits of 4 digit number
;Exit conditions: dx contains 4 digit BCD equivalent of dx
;                    and bx
;Registers altered: dx
;Subroutines called: none
;Description:
;              Upon entry to this subroutine dx and bx must
;contain the ASCII representation of a 4 digit number.
;Even if the digit to be converted is not 4 digits, these
;registers are converted and therefore the number must be
;right justified with zero fill.
;
asc_dec:
                push ax                  ;save registers
                push cx
                push bp
                push si
                mov si,000fh             ;initialize mask
                mov bp,01h               ;initialize bp
asc_dec_1:      mov al,dl                ;get first char
                and ax,si
                mov cl,al                ;save result
                mov al,dh                ;get second char
                and ax,si
                mov ch,al                ;save result
                mov ax,0                 ;clear ax
                mov al,cl
                mov cl,4
                shl ch,cl                ;shift
                add al,ch                ;result in al
                mov dx,ax                ;place in dx
                cmp bp,00h               ;check for end
                jz asc_dec_2
                mov di,dx                ;most signif in di
                mov bp,00h
                mov dx,bx                ;adj least signif
                jmp asc_dec_1
asc_dec_2:      mov cl,08h
                shl di,cl
```

```
                add dx,di                   ;final result
                pop si                      ;restore regs
                pop bp
                pop cx
                pop ax
                ret
;**************************************************************
;Subroutine:dec_bin
;Entry conditions: dx - Contains 4 digit BCD number
;Exit conditions: dx - Contains binary equivalent
;Registers Altered: dx
;Subroutines called: save,restor
;Description:
;       This subroutine converts the binary coded decimal
;(BCD) number found in dx into its binary equivalent
;and places the result in dx.
;
dec_bin:
                call save                   ;save all regs
                mov di,dx                   ;save a copy
                mov si,0                    ;init result
                mov bp,0300h                ;power of 10
                mov cl,0ch                  ;init shift factor
dec_bin_1:      mov ax,bp                   ;move power of 10
                mov ch,ah                   ;to ch
                shr dx,cl                   ;shift BCD number
                mov bx,dx                   ;move it to bx
                and bx,000fh                ;mask off the byte
dec_bin_2:      mov ax,0ah                  ;multiply factor
                mul bx
                mov bx,ax                   ;move result to bx
                dec ch                      ;dec power of 10
                jnz dec_bin_2
                add si,bx                   ;add to result
                sub bp,0100h                ;adjust power of 10
                                            ;for next loop
                mov dx,di                   ;restore number
                sub cl,04h                  ;adjust shift count
                jnz dec_bin_1
                mov dx,di                   ;restore number
                and dx,000fh                ;mask off last byte
                add si,dx                   ;final result
                mov bin_num,si              ; result 'bin_num'
                call restor
                mov dx,bin_num              ;move result to dx
                ret
;**************************************************************
;Subroutine: bin_dec
;Entry conditions: dx contains binary number in range 0-999
;Exit conditions: dx contains 4 digit BCD equivalent
;Registers altered: dx
```

```
;Subroutines called: save,restor
;Description:
;           The binary number found in register dx upon
;entry to this routine is converted to its binary coded
;decimal equivalent.  Note that no checks are made on the
;validity of the number but that any number outside of the
;range 0-999 decimal will produce unpredictable results.
;
bin_dec:
                    call save                 ;save all registers
                    mov di,dx                 ;save the number
                    mov bx,offset table_1     ;translate table
                    mov bp,01h                ;initialize mask
                    mov ans_ls,0h             ;initialize result
                    mov ans_ms,0h
                    mov cl,0h                 ;init tbl position
bin_dec_1:          and dx,bp                 ;check for bit set
                    cmp dx,0h
                    jnz bin_dec_3             ;update the result
bin_dec_2:          inc cl                    ;update position
                    cmp cl,0ah                ;test last check
                    jz bin_dec_6
                    shl bp,1                  ;update the mask
                    mov dx,di                 ;restore number
                    jmp bin_dec_1
bin_dec_3:          mov al,cl                 ;offset trans tbl
                    xlat table_1              ;translate number
                    add al,ans_ls             ;update the result
                    daa                       ;adjust result BCD
                    mov ans_ls,al             ;store result
                    jb bin_dec_5
bin_dec_4:          mov bx,offset table_2     ;point to table_2
                    mov al,cl
                    xlat table_2              ;translate number
                    add al,ans_ms
                    daa                       ;adjust to BCD
                    mov ans_ms,al             ;store result
                    mov bx,offset table_1     ;restore BX
                    jmp bin_dec_2
bin_dec_5:          mov al,ans_ms             ;add 1 to ms byte
                    add al,1h
                    daa                       ;adjust result
                    mov ans_ms,al             ;store result
                    jmp bin_dec_4
bin_dec_6:          mov ax,0
                    mov al,ans_ms             ;finalize result
                    mov cl,8                  ;load shift count
                    shl ax,cl
                    mov cx,0
                    mov cl,ans_ls
                    add ax,cx
```

```
                mov dec_num,ax          ;save result
                call restor             ;restor registers
                mov dx,dec_num          ;move result to dx
                ret
;*****************************************************************
;Subroutine: dec_asc
;Entry conditions: dx contains 4 digit BCD number
;Exit conditions: dx contains most significant 2 digits
;                    in ASCII code
;                 bx contains least significant 2 digits
;                    in ASCII code
;Registers altered: dx,bx
;Subroutines called: none
;Description:
;               The 4 digit BCD number found in dx upon
;entry is converted to its ASCII equivalent and placed
;in dx and bx.  No check is made on the validity of the
;data in dx.
;
dec_asc:
                push ax                 ;save registers
                push cx
                push di
                push bp
                mov bp,01h              ;initialize flag
                mov al,dl               ;al low nybble
                mov bl,dl               ;bl high nybble
dec_asc_1:      and al,0fh
                and bl,0f0h
                add al,30h              ;convert to ASCII
                mov cl,4
                shr bl,cl
                add bl,30h              ;convert to ASCII
                mov cl,al
                mov ch,bl               ;move result to cx
                cmp bp,00h              ;last conversion?
                je dec_asc_2
                mov bp,00h
                mov di,cx               ;low result to di
                mov al,dh
                mov bl,dh
                jmp dec_asc_1
dec_asc_2:      mov dx,cx
                mov bx,di
                pop bp                  ;restore registers
                pop di
                pop cx
                pop ax
                ret
;*****************************************************************
;Subroutine: save
```

```
;Entry conditions: none
;Exit conditions: all registers are pushed on the stack
;Registers altered: none
;Subroutines called: none
;Description:
;       This subroutine pushes all of the registers on the
;stack.  Note that the call return is preserved.
;
save:
                mov temp_ax,ax          ;save ax
                pop ax                  ;pop return address
                mov temp_stack,ax       ;save call return
                mov ax,temp_ax          ;restore ax
                push ds                 ;push all registers
                push es
                push ss
                push bp
                push si
                push di
                push ax
                push bx
                push cx
                push dx
                mov ax,temp_stack ;restore call return
                push ax
                ret
;**************************************************************
;Subroutine: restor
;Entry conditions: stack contains all the registers
;Exit conditions: registers are restored to the condition
;prior to the call to 'save'
;Registers altered: all except cs
;Subroutines called: none
;Description:
;       This subroutine returns all registers to their
;same condition prior to the call to 'save'.
;
restor:
                pop ax            ;pop return address
                mov temp_stack,ax ;save the call return
                pop dx            ;pop all registers
                pop cx
                pop bx
                pop ax
                pop di
                pop si
                pop bp
                pop ss
                pop es
                pop ds
                mov temp_ax,ax    ;save ax
```

```
                    mov ax,temp_stack ;restore call return
                    push ax
                    mov ax,temp_ax    ;restore ax
                    ret
;------------------------------------------------------------
;
;                   DATA SECTION
;
;
bin_num             dw      0                ;used by dec_bin
temp_ax             dw      0                ;used by save and
temp_stack          dw      0                ;restor
temp_tk             dw      00h              ;used by trk_num
;************************************************************
;                   STORAGE REQUIRED BY BIN_DEC
;
;
table_1             db      01h
                    db      02h
                    db      04h
                    db      08h
                    db      16h
                    db      32h
                    db      64h
                    db      28h
                    db      56h
                    db      12h
table_2             db      00h
                    db      00h
                    db      00h
                    db      00h
                    db      00h
                    db      00h
                    db      00h
                    db      01h
                    db      02h
                    db      05h
ans_ms              db      00h
ans_ls              db      00h
dec_num             dw      00h
;------------------------------------------------------------
;                   STORAGE REQUIRED BY CON_IN
;
;
buffer              db      00h
num_chars           db      00h
asc_data_1          db      00h
asc_data_2          db      00h
asc_data_3          db      00h
ms_data             dw      00h
ls_data             dw      00h
```

```
;-----------------------------------------------------------
;                  STORAGE REQUIRED BY REV
;
;
cmd_name_0      db      'Initialize the Disk $'
cmd_name_1      db      'Verify Initialization $'
cmd_name_2      db      'Initialize and Verify $'
cmd_name_3      db      'Format the Disk $'
cmd_name_4      db      'Verify the Format $'
;
rev_tabl        db      wip,cr,lf,'Command to be executed:'
                db      '$',cr,lf
;
rev_tabl_1      db      'Physical address of logical'
                db      ' sector 0: '
asc_log_sec0    db      00h
asc_log_sec0_1  db      00h
asc_log_sec0_2  db      00h
                db      cr,lf,'Sector skew factor: '
;
asc_skw_fac     db      00h
asc_skw_fac_1   db      00h
asc_skw_fac_2   db      00h
                db      cr,lf,'Location of spare sector: '
;
asc_spar        db      00h
asc_spar_1      db      00h
asc_spar_2      db      00h
rev_tabl_2      db      cr,lf,'Disk head number: '
asc_head        db      00h
asc_head_1      db      00h
asc_head_2      db      00h
                db      cr,lf,'Beginning track number: '
asc_beg_trk     db      00h
asc_beg_trk_1   db      00h
asc_beg_trk_2   db      00h
                db      cr,lf,'Ending track number: '
asc_end_trk     db      00h
asc_end_trk_1   db      00h
asc_end_trk_2   db      00h
                db      cr,lf,lf
                db      'Strike <enter> to continue'
                db      '$'
;-----------------------------------------------------------
;                  STORAGE REQUIRED BY PROC_ERR
;
;
err_code        db      00h
proc_err_tabl   db      wip,cr,lf,'Statistics on Command '
                db      'Abortion:'
                db      cr,lf,'Command being executed: $'
```

```
proc_err_tabl_1 db      cr,lf,'Disk head number: '
asc_dk_head     db      00h
asc_dk_head_1   db      00h
asc_dk_head_2   db      00h
                db      cr,lf,'Last trk number processed: '
asc_trk         db      00h
asc_trk_1       db      00h
asc_trk_2       db      00h
                db      cr,lf,'Last sec number processed: '
asc_sec         db      00h
asc_sec_1       db      00h
asc_sec_2       db      00h
                db      cr,lf,'Error code: '
asc_err_c       db      00h
asc_err_c_1     db      00h
asc_err_c_2     db      00h
                db      cr,lf,lf
                db      'Strike <enter> to continue'
                db      '$'
;--------------------------------------------------------------
;               JUMP TABLES
;
;
jmp_tabl_1      dw      in_ver_dsk
                dw      in_ver_dsk
                dw      in_ver_dsk
                dw      fm_ver_dsk
                dw      fm_ver_dsk
                dw      descr
                dw      s_end
jmp_tabl_2      dw      read_0
                dw      read_1
                dw      read_2
                dw      read_3
                dw      read_4
                dw      read_5
                dw      read_6
jmp_tabl_3      dw      rev_ent
                dw      cng_ent
                dw      e_cmmd
                dw      main
                dw      s_end
jmp_tabl_4      dw      cmd_name_0
                dw      cmd_name_1
                dw      cmd_name_2
                dw      cmd_name_3
                dw      cmd_name_4
jmp_tabl_5      dw      cng_7
                dw      cng_8
                dw      cng_9
                dw      cng_10
```

```
                dw      cng_11
                dw      cng_12
jmp_tabl_6      dw      cng_10
                dw      cng_11
                dw      cng_12
;-----------------------------------------------------------
;               DESCRIPTION OF COMMANDS
;
;
read_0          db      wip,cr,lf,'Initialize the Disk:'
                db      cr,lf,'   This command is used '
                db      'to write the address and '
                db      'data fields on the disk.'
                db      cr,lf,'It should only be used if'
                db      ' a disk fault is suspected'
                db      cr,lf,lf
                db      'strike <enter> to continue$'
read_1          db      wip,cr,lf,'Verify Initialization:'
                db      cr,lf,'  This command is used in '
                db      'conjunction with the Initialize '
                db      'command.',cr,lf,'Any errors that '
                db      'are discovered'
                db      'during verification '
                db      'are reported at the console.'
                db      cr,lf
                db      'The error codes can be found in '
                db      'the Micropolis Technical Manual '
                db      'pp 24-25'
                db      cr,lf,lf
                db      'strike <enter> to continue$'
read_2          db      cr,lf,'Initialize and '
                db      'Verify simultaneously:',cr,lf
                db      'This is a combination of the '
                db      'previous two commands.'
                db      cr,lf,lf
                db      'strike <enter> to continue$'
read_3          db      wip,cr,lf,'Format the Disk:'
                db      cr,lf
                db      'The controller will place 51h in '
                db      'all data fields',cr,lf,'during '
                db      'initialization of the disk.'
                db      cr,lf
                db      'This command is used to replace '
                db      '51h with E5h as this is ',cr,lf
                db      'what CP/M expects to find to'
                db      'create a directory '
                db      cr,lf,lf
                db      'strike <enter> to continue$'
read_4          db      wip,cr,lf,'Verify the Format:'
                db      cr,lf
                db      '   Verifies that E5h is in the '
```

```
                db      'data fields of the disk.'
                db      cr,lf,lf
                db      'strike <enter> to continue$'
read_5          db      wip,cr,lf,'Read a Description of '
                db      'these commands:',cr,lf,lf
                db      '   A quick look at the commands '
                db      'available in the Micropolis '
                db      cr,lf,'Maintenance area.',cr,lf,lf
                db      'strike <enter> to continue$'
read_6          db      wip,cr,lf,'End this session:'
                db      cr,lf
                db      '  Immediately terminates the '
                db      'session with no further action.'
                db      cr,lf,lf
                db      'strike <enter> to continue$'
;-----------------------------------------------------------
;               MENUS
;
;
menu_1          db      cr,lf,'Select Option:'
                db      cr,lf,'(0) Initialize the disk'
                db      cr,lf,'(1) Verify Initialization'
                db      cr,lf,'(2) Initialize and Verify '
                db      'simultaneously'
                db      cr,lf,'(3) Format the disk'
                db      cr,lf,'(4) Verify the Format'
                db      cr,lf,'(5) Read a description of '
                db      'these commands'
                db      cr,lf,'(6) End this session'
                db      cr,lf,'Enter selection ==> $'
menu_2          db      cr,lf,'Select Option:'
                db      cr,lf,lf,'(0) Review entrys'
                db      cr,lf,'(1) Change an entry'
                db      cr,lf,'(2) Execute command'
                db      cr,lf,'(3) Start over'
                db      cr,lf,'(4) End session'
                db      cr,lf,lf,'Enter selection ==> $'
menu_3          db      cr,lf,'Select value to change:'
                db      cr,lf,lf,'(0) Physical address of '
                db      'logical sector 0'
                db      cr,lf,'(1) Sector skew factor'
                db      cr,lf,'(2) Location of spare '
                db      cr,lf,'(3) Disk head number'
                db      cr,lf,'(4) Beginning track number'
                db      cr,lf,'(5) Ending track number'
                db      cr,lf,lf,'Enter selection ==> $'
menu_4          db      cr,lf,lf,'Select value to change:'
                db      cr,lf,'(0) Disk head number'
                db      cr,lf,'(1) Beginning track number'
                db      cr,lf,'(2) Ending track number'
                db      cr,lf,lf,'Enter selection ==> $'
```

```
;-------------------------------------------------------------
;                ERROR MESSAGES
;
micrst_err      db      cr,lf,'Micropolis Disk Reset Errors'
err_in          db      wip,be,cr,lf,'Error in input.'
                db      'Only integer data is valid. '
                db      cr,lf, 'Try again ==> $'
err_1           db      wip,be,cr,lf,'You have not '
                db      'selected a valid option. $'
err_2           db      wip,be,cr,lf,'ERROR:$'
;-------------------------------------------------------------
;                GENERAL MESSAGES
;
end_msg         db      cr,lf,'Session has been '
                db      'terminated.$'
warn            db      wxip,cr,lf,'---------------------'
                db      'W A R N I N G---------------------'
                db      cr,lf,'Use of this program will'
                db      'destroy the contents of disk !!!!'
                db      cr,lf,'Do you wish to '
                db      'continue (y/n)? $'
msg_1           db      cr,lf,'Input disk head number.'
                db      cr,lf,'Valid range is 0 to 4 ==> $'
msg_2           db      cr,lf,'Input the physical '
                db      'address of logical sector 0.'
                db      cr,lf,'Valid range  0 to 23 ==> $'
msg_3           db      cr,lf,'Input sector skew factor'
                db      cr,lf,'Valid range  0 to 23 ==> $'
msg_4           db      cr,lf,'Valid range  0 to 579 ==> $'
msg_5           db      cr,lf,'Input beginning '
                db      'trk number.$'
msg_6           db      cr,lf,'Input ending track number.$'
msg_7           db      cr,lf,'Input location of '
                db      'spare sector'
                db      cr,lf,'Valid range 0 to 255 ==> $'
msg_8           db      cr,lf,lf,'Currently formatting'
                db      ' and/or verifying format of disk'
                db      cr,lf,'Please standby......$'
msg_10          db      cr,lf,lf,'Currently initializing '
                db      'and/or verifying disk'
                db      cr,lf,'Please standby......$'
msg_11          db      cr,lf,'Command was successfully'
                db      ' executed.$'
;-------------------------------------------------------------
;                MICROPOLIS PARAMETER TABLE
;
cmd_byte        db      00h
parm1           db      00h
parm2           db      00h
parm3           db      00h
parm4           db      00h
```

```
parm5           db      00h
parm6           db      00h
go_byte         db      00h
;------------------------------------------------------------------
;               STORAGE REQUIRED BY MAIN PROGRAM
;
cmd_type        db      00h
log_sec0        db      00h
skw_fac         db      00h
spar            db      00h
head            db      00h
beg_trk_num     dw      00h
end_trk_num     dw      00h
                end
```

APPENDIX C

PROGRAM LISTING OF CPMBIOS.A86

```
;Prog Name   : CPMBIOS.A86 (Master CPM Bios)
;Date        : 5 April 1983
;Written by  : Digital Research
;Modified by : Mark L. Perry
;For         : Thesis (AEGIS Modeling Group)
;Advisor     : Professor Cotton
;Purpose     : This bios is for use with the iSB86/12A.
;            : Includes login and logout routines and all
;            : Read/Write operation conducted via common
;            : memory.  It also includes the code for
;            : generating a loader for the Remex floppy
;            : disk.

;*************************************************************
;                          EQUATES
;*************************************************************

true            equ -1
false           equ not true
cr              equ 0dh         ;carriage return
lf              equ 0ah         ;line feed
error           equ 0ffh        ;general error indication
master          equ true        ;for master/slave version
loader_bios     equ false       ;set for loader version
cmemseg         equ 0e000h      ;common memory segment

;system addresses

bdos_int        equ 224         ;reserved BDOS interrupt
;
;
        IF      not loader_bios
;-------------------------------------------------------------
;
ccp_offset      equ 0000h       ;start of CCP code
bdos_offset     equ 0B06h       ;BDOS entry point
bios_offset     equ 2500h       ;start of BIOS code
;
;-------------------------------------------------------------
        ENDIF                   ;not loader_bios

        IF      loader_bios
;-------------------------------------------------------------
```

```
;
bios_offset      equ 1200h        ;start of ldbios
ccp_offset       equ 0003h        ;base of CPMLOADER
bdos_offset      equ 0406h        ;stripped BDOS entry
;
;------------------------------------------------------------
        ENDIF                     ;loader_bios

;console via the i8251 USART

cstat            equ 0dah         ;status port
cdata            equ 0d8h         ;data port
tbemsk           equ 1            ;transmit buffer empty
rdamsk           equ 2            ;receive data available

        cseg
        org      ccpoffset
ccp:
        org      bios_offset

;************************************************************
bios:            ;JUMP VECTORS
;************************************************************

        jmp INIT        ;Enter from BOOT ROM or LOADER
        jmp WBOOT       ;Arrive here from BDOS call 0
        jmp CONST       ;return console keyboard status
        jmp CONIN       ;return console keyboard char
        jmp CONOUT      ;write char to console device
        jmp LISTOUT     ;write character to list device
        jmp PUNCH       ;write character to punch device
        jmp READER      ;return char from reader device
        jmp HOME        ;move to trk 00 on sel drive
        jmp SELDSK      ;select disk for next rd/write
        jmp SETTRK      ;set track for next rd/write
        jmp SETSEC      ;set sector for next rd/write
        jmp SETDMA      ;set offset for user buff (DMA)
        jmp READ        ;read a 128 byte sector
        jmp WRITE       ;write a 128 byte sector
        jmp LISTST      ;return list status
        jmp SECTRAN     ;xlate logical->physical sector
        jmp SETDMAB     ;set seg base for buff (DMA)
        jmp GETSEGT     ;return offset of Mem Desc Table
        jmp GETIOBF     ;return I/O map byte (iobyte)
        jmp SETIOBF     ;set I/O map byte (iobyte)

;************************************************************
;                Entry Point Routines
;************************************************************
```

```
        IF      not loader_bios
;-----------------------------------------------------------------
;
        include login.a86       ;login & logout procedures
;
;-----------------------------------------------------------------
        ENDIF

;*****************************************************************
INIT:   ;print signon message and initialize hardware
        ;and software

        mov  ax,cs              ;we entered with a JMPF
        mov  ss,ax              ;so use cs: as initial
        mov  ds,ax              ;segment values
        mov  es,ax
        mov  sp,offset stkbase  ;use local stack
        cld                     ;clear direction flag

        IF      not loader_bios
;-----------------------------------------------------------------
;
;       This is a BIOS for the CPM.SYS file
;       Setup all interrupt vectors in low
;       memory to address trap
;
        push ds
        push es
        mov iobyte,0            ;clear i/o byte
        mov  ax,0               ;address trap routine
        mov  ds,ax
        mov  es,ax
        mov  int0_offst,offset int_trap
        mov  int0_segment,cs
        mov  di,4               ;propagate to remaining
                                ;vectors
        mov  si,0
        mov  cx,510
    rep movs ax,ax
        mov  bdio,bdos_offset   ;correct bdos int vector
        pop  es
        pop  ds
;
;-----------------------------------------------------------------
        ENDIF                   ;not loader_bios

        IF      loader_bios
;-----------------------------------------------------------------
;
;       This is a BIOS for the LOADER
;
```

```
        push ds                     ;save data segment
        mov ax,0
        mov ds,ax                   ;point to segment 0
        mov bdio,bdos_offset        ;correct offset
        mov bdis,CS                 ;bdos interrupt segment
        pop ds
;
;--------------------------------------------------------------
        ENDIF                       ;loader_bios

        call con_init               ;initialize console
        xor  bx,bx                  ;get mass storage
    ini1:
        mov  ax,intbl[bx]           ;initialization table
        or   ax,ax                  ;quit if end of table
        jz   ini2
        push bx
        call ax                     ;call init entry
        pop  bx
        inc  bx                     ;step to next entry
        inc  bx
        jmp  ini1                   ;loop for next

        IF      not loader_bios
;--------------------------------------------------------------
;
    ini2:
        call login
        mov  bx,offset signon       ;print sign on msg
        call pmsg
        mov  cl,user
;
;--------------------------------------------------------------
        ENDIF                       ;not loader_bios

        IF      loader_bios
;--------------------------------------------------------------
;
    ini2:
        mov bx,offset signon1       ;print sign on message
        call pmsg
        mov cl,0                    ;default to 'a' on coldstart
        mov unit,0
;
;--------------------------------------------------------------
        ENDIF                       ;loader_bios

        jmp  ccp                    ;jump to cold entry of CCP

;**************************************************************
```

```
WBOOT:          ;enter CCP at command level

        jmp  ccp+6

;*****************************************************************
CONST:          ;return console status

        in   al,cstat
        and  al,rdamsk
        jz   con1
        or   al,0ffh              ;return non-zero if rda
con1:   ret

;*****************************************************************
CONIN:          ;get a character from console

        call CONST
        jz   CONIN                ;wait for RDA
        in   al,cdata
        and  al,7fh             ;read data & remove parity bit
        ret

;*****************************************************************
CONOUT:         ;send a character to console

        in   al,cstat
        and  al,tbemsk            ;get console status
        jz   CONOUT
        mov  al,cl
        out  cdata,al             ;xmit buff is empty
        ret

;*****************************************************************
LISTOUT:        ;send character to list device
                ;not yet implemented

        ret

;*****************************************************************
PUNCH:          ;write character to punch device
                ;not implemented

        ret

;*****************************************************************
READER:         ;get character from reader device
                ;not implemented
```

```
        mov  al,1ah                 ;return eof
        ret

;*****************************************************************
HOME:            ;move selected disk to trk 00
;
        mov  track,0
        xor  bx,bx
        mov  bl,unit
        add  bx,bx
        call hmtbl[bx]
        ret
;
;
;*****************************************************************
SELDSK:
                 ;return pointer to appropriate 'disk
                 ;parameter block' (zero for bad unit no)
                 ;NOTE: nunits is defined in the .cfg file

        mov  unit,cl                ;save unit number
        mov  bx,0000h               ;ready for error return
        cmp  cl,nunits              ;return if beyond max unit
        jnb  sel1
        mov  bl,unit
        add  bx,bx
        call dsktbl[bx]
        xor  bx,bx
        mov  bl,unit                ;bx = cl * 16
        mov  cl,4
        shl  bx,cl
        mov  cx, offset dpbase      ;bx += &dpbase
        add  bx,cx
    sel1:
        ret

;*****************************************************************
SETTRK:          ;set track address

        mov  track,cl
        xor  bx,bx
        mov  bl,unit
        add  bx,bx
        call trktbl[bx]
        ret

;*****************************************************************
SETSEC:          ;set sector number

        mov  sector,CL
```

```
        xor   bx,bx
        mov   bl,unit
        add   bx,bx
        call  sectbl[bx]
        ret

;*****************************************************************
SETDMA:             ;set DMA offset given by cx

        mov   dma_adr,cx
        ret


;*****************************************************************
READ:   ;read selected unit, track, sector to dma addr
        ;read and write operate by an indirect call
        ;through the appropriate table contained in
        ;the configuration file.  It is the programmer's
        ;responsibility to ensure that the entry points
        ;in these tables match the unit type

        xor   bx,bx
        mov   bl,unit
        add   bx,bx
        call  rdtbl[bx]
        ret


;*****************************************************************
WRITE:  ;write from dma address to selected
        ;unit, track, sector

        xor   bx,bx
        mov   bl,unit
        add   bx,bx
        call  wrtbl[bx]
        ret


;*****************************************************************
LISTST:             ;poll list device status
                    ;not implemented

        or    al,0ffh               ;return ready anyway or
        ret                         ;system may hang up


;*****************************************************************
SECTRAN:            ;translate sector cx by table at [dx]
                    ;NOTE:  this routine is not adequate for
                    ;the case of >= 256 sectors per track
```

```
                    ;still it's better than DR's which is not
                    ;adequate for the no table case either

          mov  ch,0
          mov  bx,cx
          cmp  dx,0                 ;check for no table case
          je   sel
          add  bx,dx                ;add sector to table addr
          mov  bl,[bx]              ;get logical sector
     sel:
          ret

;******************************************************************
SETDMAB:            ;set DMA segment given by cx

          mov  dma_seg,cx
          ret


;******************************************************************
GETS GT:            ;return addr of physical memory table

          mov  bx,offset segtable
          ret


;******************************************************************
GETIOBF:            ;return iobyte value
                    ;note - this function and SETIOBF
                    ;are OK but to implement the function
                    ;the character IO entry point routines
                    ;must be modified to redirect IO
                    ;depending on the value of iobyte

          mov  al,iobyte
          ret


;******************************************************************
SETIOBF:            ;set iobyte value

          mov  iobyte,cl
          ret


;******************************************************************
;                        SUBROUTINES
;******************************************************************
```

```
        IF      not loader_bios
;--------------------------------------------------------------
;
int_trap:       ;interrupt trap - non interrupt
                ;driven system so should never get
                ;here - send mesage and halt

        cli                         ;block interrupts
        mov  ax,cs
        mov  ds,ax                  ;get our data segment
        mov  bx,offset int_trp
        call pmsg
        hlt                         ;hardstop

;
;--------------------------------------------------------------
        ENDIF                       ;not loader_bios

con_init:       ;initialize console driver
                ;actually done by the iSBC86/12a monitor

        ret


;--------------------------------------------------------------
pmsg:           ;send a message to the console

        mov  al,[bx]                ;get next char from message
        test al,al
        jz   pms1                   ;if zero return
        mov  cl,al
        call CONOUT                 ;print it
        inc  bx
        jmps pmsg
    pms1:
        ret

;               DISK INCLUDE FILES
;**************************************************************

        include rxflop.a86
        include michard.a86

        IF      not loader_bios
;--------------------------------------------------------------
;
        include mb80dsk.a86
        include rxhard.a86
;
;--------------------------------------------------------------
```

```
        ENDIF                       ;not loader_bios

;*****************************************************************
;                   RESOURCE ALLOCATION
;*****************************************************************

;       low-level synchronization of access to the shared
;       device.  <sync.a86> must include the three entry
;       points defined in the cfg.files.  These are
;       called on initialization and before and after
;       accessing the resource respectively.

        IF      not loader_bios
;-----------------------------------------------------------------
;
        include sync.a86
;
;-----------------------------------------------------------------
        ENDIF                       ;not loader_bios


;*****************************************************************
;                  DATA & LOCAL STACK AREA
;*****************************************************************

        cseg $

signon  db      cr,lf,cr,lf
        db      cr,lf,lf,'                                           '
        if master
        db      'CPM/86 Master '
        endif
        if not master
        db      'CPM/86 Slave'
        endif

        IF      not loader_bios
;-----------------------------------------------------------------
;
        db      cr,lf,lf,'                         Modified '
        db      '22 April 1983 by'
        db      cr,lf,'                                 '
        db      'Mark L. Perry'
        db      cr,lf,lf
        db      '           For use with a Bubble Memory , '
        db      'the REMEX Dataware House,',cr,lf
        db      '           and the Micropolis Disk Drive'
        db      cr,lf,lf,lf,0
;
;-----------------------------------------------------------------
        ENDIF                       ;not loader_bios
```

```
        IF      loader_bios
;------------------------------------------------------------
;
signon1 db      cr,lf,'                 CP/M-86 Loader'
        db      cr,lf,'                  Version 1.0'
        db      cr,lf,'Loading CP/M from the Remex'
        db      ' Floppy Disk Drive. . .',0

;
;------------------------------------------------------------
        ENDIF                           ;loader_bios

int_trp db      cr,lf
        db      'Interrupt Trap Halt'
        db      cr,lf,0
iobyte  rb      1       ;character i/o redirection byte
unit    rb      1       ;selected unit
track   rb      1       ;selected track
sector  rb      1       ;selected sector
dma_adr rw      1       ;selected DMA address
dma_seg rw      1       ;selected DMA segment
loc_stk rw      100     ;local stack for initialization
stkbase equ     offset $

;system memory segment table

segtable        db 1                    ;1 segment
                dw tpa_seg              ;1st seg starts after BIOS
                dw tpa_len              ;and extends to top of TPA

;************************************************************
;               DISK DEVICE TABLES
;************************************************************

;the included .cfg file below maps unit number to disk
;device type.  it provides tables of entry point
;addresses for use by init, read and write.  These
;addresses must appear in the appropriate include
;file for the particular device type

        include cpmmast.cfg     ;read in configuration
                                ;table
        IF      loader_bios
;------------------------------------------------------------
;

        include ldrmast.cfg     ;read in configuration table

;
```

```
;---------------------------------------------------------------
        ENDIF                       ;loader_bios


;the included .lib file contains disk definition
;tables detailing disk characteristics for the bdos
;.lib files are generated by GENDEF from definition.
;files and must comply with the allocations made in
;the corresponding configuration file.

        IF      not loader_bios
;---------------------------------------------------------------
;
        include cpmmast.lib         ;read in disk def tables
;
;---------------------------------------------------------------
        ENDIF                       ;not loader_bios

        IF      loader_bios
;---------------------------------------------------------------
;
        include ldrmast.lib         ;read in disk def tables
                                    ;for the loader
;
;---------------------------------------------------------------
        ENDIF                       ;loader_bios


;***************************************************************
;               END OF BIOS
;***************************************************************

lastoff equ     offset $
tpa_seg equ     (lastoff+0400h+15) / 16
tpa_len equ     0fffh - tpa_seg
                db 0                ;fill last addr for GENCMD

;***************************************************************
;               PAGE ZERO TEMPLATE
;***************************************************************

                dseg    0           ;absolute low memory
                org     0           ;(interrupt vectors)
int0_offst      rw      1
int0_segment    rw      1
                rw      2*(bdos_int-1)
bdio            rw      1           ;bdos interrupt offset
bdis            rw      1           ;bdos interrupt segment
        end
```

APPENDIX D

PROGRAM LISTING OF CPMMAST.CFG

```
;Prog Name   : CPMMAST.CFG ( Master Configuration for CPM)
;Date        : 25 April 1983
;Modified by: Mark L. Perry
;For         : Thesis (AEGIS Modeling Group)
;Advisor     : Professor Cotton
;Purpose     : This code is an include file w/in CPMBIOS.A86.
;              It contains the device tables for access to
;              initialization, read, & write routines.  It
;              also allows for the development of a loader
;              BIOS.


        IF  not loader_bios
;----------------------------------------------------------------
;                   DEFINE nunits

nunits  db 12   ;total number of mass storage  units

;----------------------------------------------------------------
        ENDIF                       ;not loader_bios

;                   INITIALIZATION TABLE
;
;intbl contains a sequence of addresses of initialization
;entry points to be called by the BIOS on entry after
;a cold boot.  The sequence is terminated by a zero entry

        IF  master and not loader_bios
;----------------------------------------------------------------
;
intbl   dw offset mb80dsk_init  ;initialize Bubble
        dw offset rxflop_init   ;initialize Remex
        dw offset initsync      ;initialize sync variables
        dw offset init_login    ;initialize login
        dw offset mic_init      ;Micropolis initialize
        dw 0                    ;end of table


;----------------------------------------------------------------
        ENDIF                       ;master and not loader_bios


        IF not master and not loader_bios
```

```
;--------------------------------------------------------------
;
intbl   dw offset mb80dsk_init  ;initialize Bubble
        dw offset rxflop_init   ;initialize Remex
        dw 0                    ;end of table
;
;--------------------------------------------------------------
        ENDIF                   ;not master and not
                                ;loader_bios
;               READ TABLE

;rdtbl and wrtbl are sequences of length nunits, containing
;the addresses of the read and write entry point routines
;respectively which apply to the unit number corresponding
;to the position in the sequence.  These and the entry pts
;for initialization must correspond to those contained in
;the appropriate include files containing code specific
;to the devices.

        IF  not loader_bios
;--------------------------------------------------------------
;
rdtbl   dw offset mb80dsk_read  ;A: is a bubble memory
        dw offset rxflop_read   ;B: is Remex floppy disk 1
        dw offset rxflop_read   ;C: is Remex floppy disk 2
        dw offset rxhard_read   ;D: is Remex hard disk 0
        dw offset rxhard_read   ;E: is Remex hard disk 1
        dw offset rxhard_read   ;F: is Remex hard disk 2
        dw offset rxhard_read   ;G  is Remex hard disk 3
        dw offset mic_read      ;H: is Micropolis disk 0
        dw offset mic_read      ;I: is micropolis disk 1
        dw offset mic_read      ;J: is Micropolis disk 2
        dw offset mic_read      ;K: is Micropolis disk 3
        dw offset mic_read      ;L: is Micropolis disk 4


;--------------------------------------------------------------
;               WRITE TABLE

wrtbl   dw offset mb80dsk_write
        dw offset rxflop_write
        dw offset rxflop_write
        dw offset rxhard_write
        dw offset rxhard_write
        dw offset rxhard_write
        dw offset rxhard_write
        dw offset mic_write
        dw offset mic_write
        dw offset mic_write
        dw offset mic_write
        dw offset mic_write
```

```
;-----------------------------------------------------------------
;                        HOME TABLE
;
hmtbl   dw offset mb80dsk_home
        dw offset rxflop_home
        dw offset rxflop_home
        dw offset rxhard_home
        dw offset rxhard_home
        dw offset rxhard_home
        dw offset rxhard_home
        dw offset mic_home
        dw offset mic_home
        dw offset mic_home
        dw offset mic_home
        dw offset mic_home

;-----------------------------------------------------------------
;                        SELDSK TABLE

dsktbl  dw offset mb80dsk_seldsk
        dw offset rxflop_seldsk
        dw offset rxflop_seldsk
        dw offset rxhard_seldsk
        dw offset rxhard_seldsk
        dw offset rxhard_seldsk
        dw offset rxhard_seldsk
        dw offset mic_seldsk
        dw offset mic_seldsk
        dw offset mic_seldsk
        dw offset mic_seldsk
        dw offset mic_seldsk

;-----------------------------------------------------------------
;                        SETTRK TABLE

trktbl  dw offset mb80dsk_settrk
        dw offset rxflop_settrk
        dw offset rxflop_settrk
        dw offset rxhard_settrk
        dw offset rxhard_settrk
        dw offset rxhard_settrk
        dw offset rxhard_settrk
        dw offset mic_settrk
        dw offset mic_settrk
        dw offset mic_settrk
        dw offset mic_settrk
        dw offset mic_settrk
```

```
;-----------------------------------------------------------------
;                     SETSEC TABLE

sectbl  dw offset mb80disk_setsec
        dw offset rxflop_setsec
        dw offset rxflop_setsec
        dw offset rxhard_setsec
        dw offset rxhard_setsec
        dw offset rxhard_setsec
        dw offset rxhard_setsec
        dw offset mic_setsec
        dw offset mic_setsec
        dw offset mic_setsec
        dw offset mic_setsec
        dw offset mic_setsec

;-----------------------------------------------------------------
        ENDIF                       ;not loader_bios
```

APPENDIX E

PROGRAM LISTING OF MICHARD.A86

```
;Prog Name      : MICHARD.A86 (Micropolis Hard Disk)
;Date           : 13 April 1983
;Written by     : Mark L. Perry
;For            : Thesis (AEGIS Modeling Group)
;Advisor        : Professor Cotton
;Purpose        : This code is an include file w/in the
;                 BIOS.  It contains the hardware dependent
;                 code for the Micropolis Disk Drive
;**********************************************************************
;                             EQUATES
;**********************************************************************
;
;------------  EQUATES FOR THE 8255 PIO  -----------------
;
mic_porte          equ     0ceh               ;command port
mic_porta          equ     0c8h               ;bi-directional
mic_portb          equ     0cah               ;output port
mic_portc          equ     0cch               ;control/status
mic_mode2_0_out    equ     0c0h               ;mode for 8255
;
;-------------  EQUATES FOR THE 8253 PIT  ----------------
;
mic_mode_port      equ     0016h              ;mode for timer
mic_count_port     equ     00d0h              ;counter port
mic_mode_cntl      equ     0030h              ;mode control value
mic_lsb_value      equ     0ch                ;least sig value
mic_msb_value      equ     30h                ;most sig value
;
;-------------  EQUATES FOR THE 8259A PIC  ---------------
;
mic_icw1           equ     13h                ;control word 1
mic_icw2           equ     40h                ;control word 2
mic_icw4           equ     0fh                ;control word 4
mic_ocw1           equ     0bfh
mic_pic_port1      equ     00c0h              ;icw port
mic_pic_port2      equ     00c2h              ;ocw port
;
;----------------------------------------------------------------------
;                       MICROPOLIS EQUATES
;
mic_rstrb_on       equ     00001010b          ;read signal
mic_rstrb_off      equ     00000010b          ;read signal off
mic_wstrb_on       equ     00000110b          ;write signal
```

```
mic_wstrb_off   equ     00000010b       ;write signal off
mic_stat        equ     00000000b       ;status signal
mic_cmd         equ     00000000b       ;command signal
mic_data        equ     00000001b       ;data signal
mic_strb_on     equ     00000010b       ;input latch signal
mic_strb_off    equ     00000011b       ;latch signal off
mic_ack_on      equ     00000100b       ;output signal
mic_ack_off     equ     00000101b       ;output signal off
mic_en_sel      equ     00000010b       ;select enable
mic_stndrd      equ     00010110b       ;normal reset
mic_irdy_mask   equ     00000001b       ;input ready
mic_ordy_mask   equ     00000010b       ;output ready
mic_busy_mask   equ     00010000b       ;busy
mic_mask        equ     10100000b       ;attn or dreq
mic_attn_mask   equ     10000000b       ;attn only
mic_dreq_mask   equ     00100000b       ;dreq only
mic_cmd_mask    equ     00000011b       ;command
mic_rd_cmd      equ     04eh            ;micropolis read
mic_wr_cmd      equ     047h            ;micropolis write
;
;-------------- Sector Blocking/Deblocking ---------------
;
mic_una         equ     byte ptr [BX] ;name for byte at BX
mic_blksiz      equ     16384          ;CP/M allocation size
mic_hstsiz      equ     512            ;host disk sect size
mic_hstspt      equ     24             ;host disk sects/trk
mic_hstblk      equ     mic_hstsiz/128;CP/M sects/host buff
mic_secshf      equ     2              ;log2(mic_hstblk)
mic_cpmspt      equ     mic_hstblk * mic_hstspt
                                       ;CP/M sectors/track
mic_secmsk      equ     mic_hstblk-1   ;sector mask
mic_wrall       equ     0              ;write to allocated
mic_wrdir       equ     1              ;write to directory
mic_wrual       equ     2              ;write to unallocate

        cseg    $
;--------------------------------------------------------
;   INIT                               ;called from INIT
;
mic_init:

                IF  master and not loader_bios
;--------------------------------------------------------
;
                cli                     ;disable all
                                        ;maskable
                                        ;interrupts
                mov al,mic_mode2_0_out  ;initialize to mode
                out mic_porte,al        ;0 and 2
                mov al,mic_ack_off      ;insure acknowledge
                out mic_porte,al        ;is off
```

```
                mov al,mic_strb_off      ;insure strobe
                out mic_porte,al         ;is off
                mov al,mic_en_sel        ;set select and
                out mic_portb,al         ;enable
                mov bx,offset micrst_msg;output reset
                call pmsg
                mov cx,10                ;wait 1 second
mic_init_1:     mov ax,27777
mic_init_2:     dec ax
                jnz mic_init_2
                dec cx
                jnz mic_init_1
                call mic_status          ;get the status
                cmp al,mic_stndrd
                jz mic_init_3            ;then return
                mov bx,offset micrst_err;output error
                call pmsg
;
;    load the vector table for the interrupt handler
;
mic_init_3:
                push es                  ;want to address
                mov ax,0                 ;absolute 0
                mov es,ax
                mov ax,offset mic_int_70;interrupt number
                mov es:mic_ip_70,ax      ;store inst ptr
                mov es:mic_cs_70,cs      ;and cs value
                pop es                   ;restore extra seg
;
;    initialize the interrupt controller
;
                mov al,mic_icw1          ;control word 1
                out mic_pic_port1,al     ;output it
                mov al,mic_icw2          ;control word 2
                out mic_pic_port2,al     ;output it
                mov al,mic_icw4          ;control word 4
                out mic_pic_port2,al     ;output it
                mov al,mic_ocw1          ;set mask register
                out mic_pic_port2,al
;
;    initialize the timer and set the status byte
;
                push es                  ;save extra seg
                mov ax,cmemseg           ;to address common
                mov es,ax
                mov mic_stat_byte,0ffh   ;any non-zero val
                pop es                   ;done with status
                mov al,mic_mode_cntl     ;set mode
                out mic_mode_port,al
                mov al,mic_lsb_value     ;low count value
                out mic_count_port,al
```

```
                mov al,mic_msb_value    ;high count value
                out mic_count_port,al   ;start timer
                sti                     ;restore ints
                ret                     ;and return
;
;   now set up the interrupt handler
;
mic_int_70:
                push ax                 ;save state

;   set up local stack for interrupt handler

               mov sav_ptr,sp           ;save stk pointer
               mov sav_seg,ss           ;save segment reg
               mov sp,offset int_base   ;set local pointer
               mov ax,cs                ;set local segment
               mov ss,ax
                push es
                push bx
                push cx
                push dx
                mov ax,cmemseg          ;make common
                mov es,ax               ;addressable
                lock mov al,mic_stat_byte ;check status
                cmp al,00h              ;action needed?
                jnz mic_term_2          ;no then return
                mov al,mic_cmd_byte     ;read or write?
                and al,mic_cmd_mask
                cmp al,02               ;read?
                jz mic_read_1
                call mic_send           ;must be write
mic_wr_1:
                call mic_status         ;get status
                test al,mic_mask        ;dreq or attn?
                jz mic_wr_1             ;keep checking
                test al,mic_attn_mask   ;was it attn?
                jnz mic_term            ;yes, all done
                mov dx,512              ;set counter
                xor bx,bx               ;clear bx
mic_wr_2:       mov al,mic_buff[bx]     ;send data
                call mic_data_out
                inc bx
                dec dx
                jnz mic_wr_2
                jmp mic_wr_1            ;check status
                                        ;for final result
mic_read_1:
                call mic_send           ;send command
mic_read_2:     call mic_status         ;get status
                test al,mic_mask        ;dreq or attn?
                jz mic_read_2
```

```
                test al,mic_attn_mask       ;was it attn?
                jnz mic_term                ;yes,all done
                mov dx,512                  ;must be dreq
                xor bx,bx                   ;clear bx
mic_read_3:     call mic_data_in            ;get data
                mov mic_buff[bx],al         ;store it in buffer
                inc bx
                dec dx
                jnz mic_read_3              ;continue
                jmp mic_read_2              ;get status
mic_term:
                call mic_busy               ;wait on cntrl
                call mic_irdy
                call mic_data_in            ;get termination
                and al,0fh                  ;lower 4 only
                cmp al,00h                  ;success?
                jz mic_term_1
                mov mic_stat_byte,0ffh      ;indicate failure
                jmp mic_term_2
mic_term_1:
                mov mic_stat_byte,0ah       ;indicate success
mic_term_2:
                pop dx                      ;restore all regs
                pop cx
                pop bx
                pop es

;   restore old stack segment and pointer

                mov sp,sav_ptr
                mov ax,sav_seg
                mov ss,ax
;
;   before final pop of ax reload counter
;
                mov al,mic_lsb_value        ;least sig value
                out mic_count_port,al
                mov al,mic_msb_value
                out mic_count_port,al       ;counter starts
                pop ax
                iret
;-----------------------------------------------------------
                ENDIF                        ;master and not
                                             ;loader_bios

                IF not master
;-----------------------------------------------------------
;                                            ;no special action
                ret
;-----------------------------------------------------------
                ENDIF
```

```
                IF  not loader_bios
;-----------------------------------------------------------
;HOME                                  entered from HOME jump
        ;home the selected disk
mic_home:
        mov al,mic_hstwrt        ;check for pending write
        test al,al
        jnz mic_homed
        mov mic_hstact,0         ;clear host active flag
mic_homed:
        ret

;-----------------------------------------------------------
;  SELECT DISK                     entered from SELDSK jump
mic_seldsk:
        ;select disk
        mov cl,unit
        mov mic_sekdsk,cl
        ;is this the first activation of the drive?
        test DL,1                ;lsb = 0?
        jnz mic_selset
        ;this is the first activation, clear host buff
        mov mic_hstact,0
        mov mic_unacnt,0
mic_selset:
        ret

;-----------------------------------------------------------
;   SELECT TRACK                   entered after SETTRK jump

mic_settrk:
        ;set track given by registers CX
        mov mic_sektrk,CX                   ;track to seek
        ret

;-----------------------------------------------------------
;   SELECT SECTOR                  entered after SETSEC jump

mic_setsec:
        ;set sector given by register cl
        mov mic_seksec,cl                   ;sector to seek
        ret

;-----------------------------------------------------------
;   READ                                 entered after READ jump

mic_read:
        ;read the selected CP/M sector
        mov mic_unacnt,0                    ;clear unallocated
```

```
        mov mic_readop,1                    ;read operation
        mov mic_rsflag,1                    ;must read data
        mov mic_wrtype,mic_wrual            ;treat as unalloc
        jmp mic_rwoper                      ;perform the read

;-----------------------------------------------------------
;   WRITE                          entered after WRITE jump

mic_write:
        ;write the selected CP/M sector
        mov mic_readop,Ø                    ;write operation
        mov mic_wrtype,cl
        cmp cl,mic_wrual                    ;write unallocated?
        jnz mic_cnkuna                      ;check for unalloc
;
;       write to unallocated, set parameters
;
        mov mic_unacnt,(mic_blksiz/128) ;next unalloc recs
        mov al,mic_sekdsk            ;disk to seek
        mov mic_unadsk,al            ;mic_unadsk = mic_sekdsk
        mov ax,mic_sektrk
        mov mic_unatrk,ax            ;mic_unatrk = mic_sektrk
        mov al,mic_seksec
        mov mic_unasec,al            ;mic_unasec = mic_seksec
;
;--------- Sector Block/Deblock Subroutines --------------

mic_cnkuna:
        ;check for write to unallocated sector
;
        mov bx,offset mic_unacnt            ;point "UNA"
                                            ;at UNACNT
        mov al,mic_una ! test al,al         ;any unallc remain?
        jz mic_alloc                        ;skip if not
;
;       more unallocated records remain
        dec al                              ;mic_unacnt
                                            ;= mic_unacnt-1
        mov mic_una,al
        mov al,mic_sekdsk                   ;same disk?
        mov BX,offset mic_unadsk
        cmp al,mic_una                      ;mic_sekdsk
                                            ;= mic_unadsk?
        jnz mic_alloc                       ;skip if not
;
;       disks are the same
        mov AX, mic_unatrk
        cmp AX, mic_sektrk
        jnz mic_alloc                       ;skip if not
;
```

```
;       tracks are the same
        mov al,mic_seksec                   ;same sector?
;
        mov BX,offset mic_unasec            ;point una
                                            ;at mic_unasec
;
        cmp al,mic_una                      ;mic_seksec
                                            ;= mic_unasec?
        jnz mic_alloc                       ;skip if not
;
;       match, move to next sector for future ref
        inc mic_una                         ;mic_unasec
                                            ;= mic_unasec+1
        mov al,mic_una                      ;end of track?
        cmp al,mic_cpmspt                   ;count CP/M sectors
        jb mic_noovf                        ;skip if below
;
;       overflow to next track
        mov mic_una,0                       ;mic_unasec = 0
        inc mic_unatrk                      ;mic_unatrk
                                            ;= mic_unatrk+1
;
mic_noovf:
        ;match found, mark as unnecessary read
        mov mic_rsflag,0                    ;mic_rsflag = 0
        jmps mic_rwoper                     ;perform write
;
mic_alloc:
        ;not an unallocated record, requires pre-read
        mov mic_unacnt,0                    ;mic_unacnt = 0
        mov mic_rsflag,1                    ;mic_rsflag = 1
                                            ;drop through
                                            ;to rwoper

;       Common code for READ and WRITE follows

mic_rwoper:
            ;enter here to perform the read/write
        mov mic_erflag,0                    ;no errors (yet)
        mov al, mic_seksec                  ;compute host sector
        mov cl, mic_secshf
        shr al,cl
        mov mic_sekhst,al                   ;host sect to seek

            ;active host sector?
        mov al,1
        xchg al,mic_hstact                  ;always becomes 1
        test al,al                          ;was it already?
        jz mic_filhst                       ;fill host if not

            ;host buffer active, same as seek buffer?
```

```
        mov al,mic_sekdsk
        cmp al,mic_hstdsk                   ;mic_sekdsk
                                            ;= mic_hstdsk?
        jnz mic_nomatch

            ;same disk, same track?
        mov ax,mic_hsttrk
        cmp ax,mic_sektrk                   ;host track same
                                            ;as seek track
        jnz mic_nomatch

            ;same disk, same track, same buffer?
        mov al,mic_sekhst
        cmp al,mic_hstsec                   ;mic_sekhst
                                            ;= mic_hstsec?
        jz mic_match                        ;skip if match
mic_nomatch:
            ;proper disk, but not correct sector
        mov al, mic_hstwrt
        test al,al                          ;"dirty" buffer ?
        jz mic_filhst                       ;no, don't write
        call mic_writehst                   ;yes, clear
                                            ;host buff
;
mic_filhst:
            ;may have to fill the host buffer
        mov al,mic_sekdsk ! mov mic_hstdsk,al
        mov ax,mic_sektrk ! mov mic_hsttrk,ax
        mov al,mic_sekhst ! mov mic_hstsec,al
        mov al,mic_rsflag
        test al,al                              ;need to read?
        jz mic_filhst1
;
        call mic_readhst                        ;yes, if 1
;
mic_filhst1:
        mov mic_hstwrt,0                        ;no pending wrt

mic_match:
    ;copy data to or from buffer depending on "mic_readop"
        mov al,mic_seksec                   ;mask buffer number
        and ax,mic_secmsk
        mov cl, 7 ! shl ax,cl               ;shift left 7
                                            ;(* 128 = 2**7)
;
;       ax has relative host buffer offset
;
        add ax,offset mic_hstbuf            ;ax has buff addr
        mov si,ax                           ;put in si reg
        mov di,dma_adr                      ;user buffer is
                                            ;dest if readop
```

```
        push DS ! push ES                       ;save seg regs
;
        mov ES,dma_seg                          ;set destseg
                                                ;to the users seg
                                                ;SI/DI and DS/ES
                                                ;is swapped
                                                ;if write op
        mov cx,128/2                            ;length of move
        mov al,mic_readop
        test al,al                              ;which way?
        jnz     mic_rwmove                      ;skip if read
;
;       write operation, mark and switch direction
        mov mic_nstwrt,1                        ;mic_nstwrt = 1
                                                ;(dirty buffer now)
        xchg si,di                              ;source/dest swap
        mov ax,DS
        mov ES,ax
        mov DS,dma_seg                          ;setup DS,ES
;
mic_rwmove:
        cld ! rep movs AX,AX                    ;move 16 bit words
        pop ES ! pop DS                         ;restore seg regs
;
;       data has been moved to/from host buffer
        cmp mic_wrtype,mic_wrdir                ;write directory?
        mov al,mic_erflag                       ;in case of errors
        jnz mic_return_rw                       ;no processing
;
;       clear host buffer for directory write
        test al,al                              ;errors?
        jnz mic_return_rw                       ;skip if so
        mov mic_nstwrt,0                        ;buffer written
        call mic_writenst
        mov al,mic_erflag
mic_return_rw:
        ret

;*****************************************************************
;                  MICROPOLIS HARD DISK SUBROUTINES
;*****************************************************************


mic_readnst:
        mov mic_dir,0                   ;indicate read or write
;----------------------------------------------------------------
        ENDIF                           ;not loader_bios


        IF  master and not loader_bios
;----------------------------------------------------------------
```

```
        cli                             ;clear int to be sure
        call get_common                 ;get resource

;-----------------------------------------------------------------
        ENDIF                           ;master and not
                                        ;loader_bios

        IF  not master and not loader_bios
;-----------------------------------------------------------------

        call request                    ;get resource

;-----------------------------------------------------------------
        ENDIF                           ;not master and not
                                        ;loader

        IF  not loader_bios
;-----------------------------------------------------------------

        mov bl,mic_rd_cmd                     ;set up read cmd
        call mic_set                          ;set up parameters
        call mic_trans                        ;transmit them
        call mic_trans_buff                   ;get the buffer
        call release                          ;release resource
        mov al,mic_result
        mov mic_erflag,al                     ;establish error
;-----------------------------------------------------------------
        ENDIF                                 ;not loader_bios


        IF  master and not loader_bios
;-----------------------------------------------------------------
        sti                                   ;restore int
;-----------------------------------------------------------------
        ENDIF                                 ;master and not
                                              ;loader

        IF  not loader_bios
;-----------------------------------------------------------------

        ret

;-----------------------------------------------------------------

mic_writehst:
        mov mic_dir,1                         ;indicate rd/wrt
        mov al,mic_hst_dsk                    ;ck for valid wrt
        cmp al,user
        jnz mic_wrt_err                       ;indicate error
```

```
;-----------------------------------------------------------------
        ENDIF                                   ;not loader_bios

        IF  master and not loader_bios
;-----------------------------------------------------------------

        cli                                     ;clear to be sure
        call get_common                         ;get resource

;-----------------------------------------------------------------
        ENDIF                                   ;master and not
                                                ;loader

        IF  not master and not loader_bios
;-----------------------------------------------------------------

        call request                            ;get resource

;-----------------------------------------------------------------
        ENDIF                                   ;not master and not
                                                ;loader

        IF  not loader_bios
;-----------------------------------------------------------------

        mov bl,mic_wr_cmd                       ;set up write cmd
        call mic_set                            ;set up parameters
        call mic_trans_buff                     ;transmit buffer
        call mic_trans                         ;transmit parameters
        call release                            ;release resource
        mov al,mic_result
        mov mic_erflag,al
        jmp mic_wrt_ret
mic_wrt_err:
        mov bx,offset mic_wrt_msg
        call pmsg                              ;error message
        mov mic_erflag,0ffh                    ;indicate the error
mic_wrt_ret:

;-----------------------------------------------------------------
        ENDIF                                   ;not loader_bios

        IF  master and not loader_bios
;-----------------------------------------------------------------
        sti                                     ;restore int
;-----------------------------------------------------------------
        ENDIF                                   ;master and not
                                                ;loader
```

```
        IF  not loader_bios
;-----------------------------------------------------------------

        ret

;-----------------------------------------------------------------

mic_set:
        push es
        push ax
        push cx
        push bx
        mov ax,cmemseg                          ;make common addr
        mov es,ax
        mov mic_cmd_byte,bl                     ;type of command
        mov bl,mic_hst_dsk                      ;adj for head num
        sub bl,7
        mov cl,4
        shl bl,cl
        mov mic_parm1,bl                        ;unit and head set
        mov cx,mic_hst_trk
        mov mic_parm2,cl
        mov mic_parm3,ch                        ;trk now set
        mov cl,mic_hst_sec
        mov mic_parm4,cl                        ;sector set
        mov mic_parm5,1                         ;only one to process
        mov mic_parm6,Ø                         ;not used
        mov mic_go_byte,                        ;set go byte
        pop bx                                  ;restore regs
        pop cx
        pop ax
        pop es
        ret

;-----------------------------------------------------------------

mic_trans:
        push es                                 ;save regs
        push ax
        mov ax,cmemseg                          ;make common addr
        mov es,ax
        mov mic_stat_byte,Ø                     ;indicate ready

;-----------------------------------------------------------------
        ENDIF                                   ;not loader_bios

        IF  master and not loader_bios
;-----------------------------------------------------------------

        int 7Ø                                  ;force interrupt
```

```
;------------------------------------------------------------
        ENDIF                               ;master and not
                                            ;loader

        IF  not loader_bios
;------------------------------------------------------------

mic_trans_1:
        mov al,mic_stat_byte                ;get status
        cmp al,0                            ;done?
        jz mic_trans_1
        cmp al,0ah                          ;success?
        jz mic_success_write
        mov mic_result,0ffh                 ;indicate failure
        jmp mic_fail_write
mic_success_write:
        mov mic_result,00h
mic_fail_write:
        pop ax
        pop es
        ret

;------------------------------------------------------------

mic_trans_buff:
        push es                             ;save regs
        push ds
        mov ax,cs
        mov es,ax
        mov di,offset mic_hstbuf
        mov ax,cmemseg
        mov ds,ax
        mov si,5100h
        mov cx,256
        cmp mic_dir,0
        jz mic_trans_buff_1
        xchg si,di
        mov ax,ds
        mov es,ax
        mov ax,cs
        mov ds,ax
mic_trans_buff_1:
        cld
        rep movs ax,ax
        pop ds
        pop es
        ret

;------------------------------------------------------------
        ENDIF                               ;not loader_bios
```

```
                                        ;routines are only used
    IF MASTER and not loader_bios       ;by interrupt handler
;-----------------------------------------------------------
;Subroutine: mic_status
;Entry Conditions: none
;Exit Conditions: al contains status of disk
;Registers altered: al
;Subroutines called: none
;Description:
;       This subroutine reads and returns the current
;value of the Micropolis disk controller's status port.
;
mic_status:
                mov al,mic_stat         ;enable status line
                out mic_porte,al
                mov al,mic_rstrb_on     ;turn on read
                out mic_portb,al
                mov al,mic_strb_on      ;latch the status
                out mic_porte,al
                mov al,mic_strb_off
                out mic_porte,al
                mov al,mic_rstrb_off    ;turn off read
                out mic_portb,al
                in al,mic_porta         ;bring in status
                ret

;-----------------------------------------------------------
;Subroutine: mic_send
;Entry conditions: parameters are calculated and in
;                  the byte variables
;Exit conditions: parameters and command have been sent
;Registers altered: none
;Subroutines called: mic_busy,mic_ordy,
;                    mic_irdy,mic_cmd_out,mic_data_out
;Description:
;               The command byte, six parameter bytes
;and the go byte found in the data area are sent to
;the disk controller.
;
mic_send:
                call mic_busy           ;wait for cntrl
                call mic_ordy
                call mic_cmd_out        ;send out cmd
                xor bx,bx               ;clear bx
                mov dl,7                ;set count value
mic_send_1:     call mic_busy           ;wait for cntrl
                call mic_ordy
                mov al,mic_parm1[bx]    ;get parm
                call mic_data_out       ;send it
```

```
        inc bx
        dec al                  ;done?
        jnz mic_send_1          ;get another
        ret

;-------------------------------------------------------------
;Subroutine: mic_cmd_out
;Entry conditions: 'ordy' signal has been issued by the
;                  disk controller and 'cmd_byte'
;                  contains the command to be sent.
;Exit conditions: none
;Registers altered: none
;Subroutines called: none
;Description:
;       The command in the byte variable 'cmd_byte'
;is sent to the disk controller.
;
mic_cmd_out:
        push ax                 ;save ax
        mov al,mic_cmd_byte
        out mic_porta,al        ;to bi-directional
        mov al,mic_cmd          ;enable cmd line
        out mic_porte,al
        mov al,mic_ack_on       ;activate output
        out mic_porte,al        ;buffer
        mov al,mic_wstrb_on     ;pulse the write
        out mic_portb,al        ;strobe
        mov al,mic_wstrb_off
        out mic_portb,al
        mov al,mic_ack_off      ;de-activate the
        out mic_porte,al        ;output buffer
        pop ax
        ret

;-------------------------------------------------------------
;Subroutine: mic_data_out
;Entry conditions: 'ordy' signal has been issued by the
;                  disk controller and al contains value
;                  to be sent.
;Exit conditions: none
;Registers altered: none
;Subroutines called: none
;Description:
;               A byte of data is output to the Micropolis
;disk unit.
;
mic_data_out:
        push ax                 ;save ax
        out mic_porta,al        ;to bi-directional
        mov al,mic_data         ;enable data line
        out mic_porte,al
```

```
                    mov al,mic_ack_on          ;activate output
                    out mic_porte,al           ;buffer
                    mov al,mic_wstrb_on        ;pulse the write
                    out mic_portb,al           ;strobe
                    mov al,mic_wstrb_off
                    out mic_portb,al
                    mov al,mic_ack_off         ;de-activate
                    out mic_porte,al           ;output buffer
                    pop ax                     ;restore value
                    ret
;-----------------------------------------------------------
;Subroutine: mic_data_in
;Entry conditions: 'irdy' signal has been issued by the
;                   disk controller
;Exit conditions: al contains data byte
;Registers altered: al
;Subroutines called: none
;Description:
;                   A byte of data is input from the Micropolis
;disk unit.
;
mic_data_in:
                    mov al,mic_data            ;enable data line
                    out mic_porte,al
                    mov al,mic_rstrb_on        ;turn the read
                    out mic_portb,al           ;on
                    mov al,mic_strb_on         ;latch the data
                    out mic_porte,al
                    mov al,mic_strb_off
                    out mic_porte,al
                    mov al,mic_rstrb_off       ;turn off the
                    out mic_portb,al           ;read signal
                    in al,mic_porta            ;bring in data
                    ret

;-----------------------------------------------------------
;Subroutine: mic_busy
;Entry conditions: none
;Exit conditions: disk controller has issued 'not busy'
;                   signal
;Registers altered: none
;Subroutines called: mic_status
;Description:
;                   The executing program will wait here
;until the disk controller issues the 'not busy' signal.
;
mic_busy:
                    push ax                    ;save ax
mic_busy_1:         call mic_status            ;get status
                    test al,mic_busy_mask      ;busy?
                    jz mic_busy_1
```

```
                pop ax
                ret

;------------------------------------------------------------
;Subroutine:  mic_irdy
;Entry conditions: none
;Exit conditions: disk controller has issued 'irdy'
;                  signal
;Registers altered: none
;Subroutines called: mic_status
;Description:
;               The execution of the program will
;wait here until 'irdy' is issued by the controller.
;
mic_irdy:
                push ax                     ;save ax
mic_irdy_1:     call mic_status             ;get status
                test al,mic_irdy_mask       ;ready?
                jz mic_irdy_1
                pop ax                      ;restore ax
                ret                         ;ready now

;------------------------------------------------------------
;Subroutine: mic_ordy
;Entry conditions: none
;Exit conditions: disk controller has issued the 'ordy'
;                  signal
;Registers altered: none
;Subroutines called: mic_status
;Description:
;               The execution of the program will wait
;here until 'ordy' is issued by the controller.
;
mic_ordy:
                push ax                     ;save ax
mic_ordy_1:     call mic_status             ;get status
                test al,mic_ordy_mask       ;ready?
                jz mic_ordy_1               ;not yet
                pop ax
                ret

;------------------------------------------------------------
;Subroutine: get_common

get_common:
                push es
                push cx
                mov ax,cmemseg              ;make common
                mov es,ax                   ;memory addressable
                call ticket                 ;get ticket number
                cmp bx,server               ;if ticket=server
```

```
                je get_common_4             ;then done
                mov cx,dcount               ;delay here
get_common_2:
                dec cx
                jnz get_common_2

;check Micropolis status byte

                cmp mic_stat_byte,00h
                je get_common_3
                cmp bx,server
                je get_common_4
                mov cx,dcount
                jmp get_common_2

;a status byte of 0 needs an interrupt

get_common_3:
                int 70                      ;execute int
                mov cx,dcount
                jmp get_common_2

get_common_4:
                pop cx
                pop es
                ret

;------------------------------------------------------------
        ENDIF                               ;end of routines
                                            ;used by interrupt

        IF  not loader_bios
;------------------------------------------------------------

;-----------  Micropolis Interface Packet  -------------

        eseg
        org 118h
mic_ip_70       rw  1
mic_cs_70       rw  1

        org 5000h
mic_stat_byte   rb  1                       ;status byte
mic_cmd_byte    rb  1                       ;command code
mic_parm1       rb  1                       ;parameters
mic_parm2       rb  1
mic_parm3       rb  1
mic_parm4       rb  1
mic_parm5       rb  1
mic_parm6       rb  1
mic_go_byte     rb  1
```

```
        org 5100h
mic_buff        rb 512                          ;buffer for data

;-------------  Misc Variables  --------------------------

        cseg    $

micrst_msg      db      cr,lf,'Micropolis Controller '
                db      'Resetting...',0
micrst_err      db      cr,lf,'Micropolis Disk Reset Error'
                db      cr,lf,'Use of Drives H - L will '
                db      'produce unpredictable results'
                db      cr,lf,lf,0
Mic_wrt_msg     db      cr,lf,'Write Access not '
                db      'Permitted on This '
                db      'Drive',0

;  space for local stack

int_stack       rw      100
int_base        equ     offset $
sav_ptr         rw      1
sav_seg         rw      1

mic_dir         rb      1               ;rd/wrt direction
mic_result      rb      1               ;result of rd/wrt
mic_sek_dsk     rb      1               ;seek disk number
mic_sek_trk     rw      1               ;seek track number
mic_sek_sec     rb      1               ;seek sector number
mic_hst_dsk     rb      1               ;host disk number
mic_hst_trk     rw      1               ;host track number
mic_hst_sec     rb      1               ;host sector number
mic_sek_hst     rb      1               ;seek shr micsecshf
mic_hst_act     rb      1               ;host active flag
mic_hst_wrt     rb      1               ;host written flag
mic_una_cnt     rb      1               ;unalloc rec cnt
mic_una_dsk     rb      1               ;last unalloc disk
mic_una_trk     rw      1               ;last unalloc track
mic_una_sec     rb      1               ;last unalloc sect
mic_erflag      rb      1               ;error reporting
mic_rsflag      rb      1               ;read sector flag
mic_readop      rb      1               ;1 if rd operation
mic_wrtype      rb      1               ;wrt operation type
mic_hstbuf      rb      mic_hstsiz      ;host buffer
;--------------------------------------------------------
          ENDIF
```

APPENDIX F

PROGRAM LISTING OF CPMMAST.DEF

The following disk definition statements were used with the GENDEF facility to generate the disk parameter tables.

```
disks 12
diskdef 0,1,26,0,1024,71,32,0,2
diskdef 1,1,26,6,1024,243,64,64,2
diskdef 2,1
diskdef 3,1,156,0,16384,275,128,0,1
diskdef 4,3
diskdef 5,3
diskdef 6,3
diskdef 7,0,95,0,16384,435,256,0,0
diskdef 8,7
diskdef 9,7
diskdef 10,7
diskdef 11,7
endef
```

APPENDIX G

PROGRAM LISTING OF CPMMAST.LIB

The following CPMMAST.LIB file is created by the GENDEF utility when the CPMMAST.DEF is used as the source file.

```
;                       DISKS 12
dpbase   equ            $                ;Base of Disk Parameter Blocks
dpe0     dw             xlt0,0000h       ;Translate Table
         dw             0000h,0000h      ;Scratch Area
         dw             dirbuf,dpb0      ;Dir Buff, Parm Block
         dw             csv0,alv0        ;Check, Alloc Vectors
dpe1     dw             xlt1,0000h       ;Translate Table
         dw             0000h,0000h      ;Scratch Area
         dw             dirbuf,dpb1      ;Dir Buff, Parm Block
         dw             csv1,alv1        ;Check, Alloc Vectors
dpe2     dw             xlt2,0000h       ;Translate Table
         dw             0000h,0000h      ;Scratch Area
         dw             dirbuf,dpb2      ;Dir Buff, Parm Block
         dw             csv2,alv2        ;Check, Alloc Vectors
dpe3     dw             xlt3,0000h       ;Translate Table
         dw             0000h,0000h      ;Scratch Area
         dw             dirbuf,dpb3      ;Dir Buff, Parm Block
         dw             csv3,alv3        ;Check, Alloc Vectors
dpe4     dw             xlt4,0000h       ;Translate Table
         dw             0000h,0000h      ;Scratch Area
         dw             dirbuf,dpb4      ;Dir Buff, Parm Block
         dw             csv4,alv4        ;Check, Alloc Vectors
dpe5     dw             xlt5,0000h       ;Translate Table
         dw             0000h,0000h      ;Scratch Area
         dw             dirbuf,dpb5      ;Dir Buff, Parm Block
         dw             csv5,alv5        ;Check, Alloc Vectors
dpe6     dw             xlt6,0000h       ;Translate Table
         dw             0000h,0000h      ;Scratch Area
         dw             dirbuf,dpb6      ;Dir Buff, Parm Block
         dw             csv6,alv6        ;Check, Alloc Vectors
dpe7     dw             xlt7,0000h       ;Translate Table
         dw             0000h,0000h      ;Scratch Area
         dw             dirbuf,dpb7      ;Dir Buff, Parm Block
         dw             csv7,alv7        ;Check, Alloc Vectors
dpe8     dw             xlt8,0000h       ;Translate Table
         dw             0000h,0000h      ;Scratch Area
         dw             dirbuf,dpb8      ;Dir Buff, Parm Block
```

```
        dw      csv8,alv8          ;Check, Alloc Vectors
dpe9    dw      xlt9,0000h         ;Translate Table
        dw      0000h,0000h        ;Scratch Area
        dw      dirbuf,dpb9        ;Dir Buff, Parm Block
        dw      csv9,alv9          ;Check, Alloc Vectors
dpe10   dw      xlt10,0000h        ;Translate Table
        dw      0000h,0000h        ;Scratch Area
        dw      dirbuf,dpb10       ;Dir Buff, Parm Block
        dw      csv10,alv10        ;Check, Alloc Vectors
dpe11   dw      xlt11,0000h        ;Translate Table
        dw      0000h,0000h        ;Scratch Area
        dw      dirbuf,dpb11       ;Dir Buff, Parm Block
        dw      csv11,alv11        ;Check, Alloc Vectors
;               DISKDEF 0,1,26,0,1024,71,32,0,2
dpb0    equ     offset $           ;Disk Parameter Block
        dw      26                 ;Sectors Per Track
        db      3                  ;Block Shift
        db      7                  ;Block Mask
        db      0                  ;Extnt Mask
        dw      70                 ;Disk Size - 1
        dw      31                 ;Directory Max
        db      128                ;Alloc0
        db      0                  ;Alloc1
        dw      0                  ;Check Size
        dw      2                  ;Offset
xlt0    equ     offset $           ;Translate Table
        db      1,2,3,4
        db      5,6,7,8
        db      9,10,11,12
        db      13,14,15,16
        db      17,18,19,20
        db      21,22,23,24
        db      25,26
als0    equ     9                  ;Allocation Vector Size
css0    equ     0                  ;Check Vector Size
;               DISKDEF 1,1,26,6,1024,243,64,64,2
dpb1    equ     offset $           ;Disk Parameter Block
        dw      26                 ;Sectors Per Track
        db      3                  ;Block Shift
        db      7                  ;Block Mask
        db      0                  ;Extnt Mask
        dw      242                ;Disk Size - 1
        dw      63                 ;Directory Max
        db      192                ;Alloc0
        db      0                  ;Alloc1
        dw      16                 ;Check Size
        dw      2                  ;Offset
xlt1    equ     offset $           ;Translate Table
        db      1,7,13,19
        db      25,5,11,17
        db      23,3,9,15
```

```
        db      21,2,8,14
        db      20,26,6,12
        db      18,24,4,10
        db      16,22
als1    equ     31                  ;Allocation Vector Size
css1    equ     16                  ;Check Vector Size
;               DISKDEF 2,1
dpb2    equ     dpb1                ;Equivalent Parameters
als2    equ     als1                ;Same Allocation Vector Size
css2    equ     css1                ;Same Checksum Vector Size
xlt2    equ     xlt1                ;Same Translate Table
;               DISKDEF 3,1,156,0,16384,275,128,0,1
dpb3    equ     offset $            ;Disk Parameter Block
        dw      156                 ;Sectors Per Track
        db      7                   ;Block Shift
        db      127                 ;Block Mask
        db      7                   ;Extnt Mask
        dw      274                 ;Disk Size - 1
        dw      127                 ;Directory Max
        db      128                 ;Alloc0
        db      0                   ;Alloc1
        dw      0                   ;Check Size
        dw      1                   ;Offset
xlt3    equ     offset $            ;Translate Table
        db      1,2,3,4
        db      5,6,7,8
        db      9,10,11,12
        db      13,14,15,16
        db      17,18,19,20
        db      21,22,23,24
        db      25,26,27,28
        db      29,30,31,32
        db      33,34,35,36
        db      37,38,39,40
        db      41,42,43,44
        db      45,46,47,48
        db      49,50,51,52
        db      53,54,55,56
        db      57,58,59,60
        db      61,62,63,64
        db      65,66,67,68
        db      69,70,71,72
        db      73,74,75,76
        db      77,78,79,80
        db      81,82,83,84
        db      85,86,87,88
        db      89,90,91,92
        db      93,94,95,96
        db      97,98,99,100
        db      101,102,103,104
        db      105,106,107,108
```

```
        db      109,110,111,112
        db      113,114,115,116
        db      117,118,119,120
        db      121,122,123,124
        db      125,126,127,128
        db      129,130,131,132
        db      133,134,135,136
        db      137,138,139,140
        db      141,142,143,144
        db      145,146,147,148
        db      149,150,151,152
        db      153,154,155,156
als3    equ     35                  ;Allocation Vector Size
css3    equ     0                   ;Check Vector Size
;               DISKDEF 4,3
dpb4    equ     dpb3                ;Equivalent Parameters
als4    equ     als3                ;Same Allocation Vector Size
css4    equ     css3                ;Same Checksum Vector Size
xlt4    equ     xlt3                ;Same Translate Table
;               DISKDEF 5,3
dpb5    equ     dpb3                ;Equivalent Parameters
als5    equ     als3                ;Same Allocation Vector Size
css5    equ     css3                ;Same Checksum Vector Size
xlt5    equ     xlt3                ;Same Translate Table
;               DISKDEF 6,3
dpb6    equ     dpb3                ;Equivalent Parameters
als6    equ     als3                ;Same Allocation Vector Size
css6    equ     css3                ;Same Checksum Vector Size
xlt6    equ     xlt3                ;Same Translate Table
;               DISKDEF 7,0,95,0,16384,435,256,0,0
dpb7    equ     offset $            ;Disk Parameter Block
        dw      96                  ;Sectors Per Track
        db      7                   ;Block Shift
        db      127                 ;Block Mask
        db      7                   ;Extnt Mask
        dw      434                 ;Disk Size - 1
        dw      255                 ;Directory Max
        db      128                 ;Alloc0
        db      0                   ;Alloc1
        dw      0                   ;Check Size
        dw      0                   ;Offset
xlt7    equ     offset $            ;Translate Table
        db      0,1,2,3
        db      4,5,6,7
        db      8,9,10,11
        db      12,13,14,15
        db      16,17,18,19
        db      20,21,22,23
        db      24,25,26,27
        db      28,29,30,31
        db      32,33,34,35
```

```
        db      36,37,38,39
        db      40,41,42,43
        db      44,45,46,47
        db      48,49,50,51
        db      52,53,54,55
        db      56,57,58,59
        db      60,61,62,63
        db      64,65,66,67
        db      68,69,70,71
        db      72,73,74,75
        db      76,77,78,79
        db      80,81,82,83
        db      84,85,86,87
        db      88,89,90,91
        db      92,93,94,95
als7    equ     55              ;Allocation Vector Size
css7    equ     0               ;Check Vector Size
;               DISKDEF 8,7
dpb8    equ     dpb7            ;Equivalent Parameters
als8    equ     als7            ;Same Allocation Vectr Size
css8    equ     css7            ;Same Checksum Vector Size
xlt8    equ     xlt7            ;Same Translate Table
;               DISKDEF 9,7
dpb9    equ     dpb7            ;Equivalent Parameters
als9    equ     als7            ;Same Allocation Vectr Size
css9    equ     css7            ;Same Checksum Vector Size
xlt9    equ     xlt7            ;Same Translate Table
;               DISKDEF 10,7
dpb10   equ     dpb7            ;Equivalent Parameters
als10   equ     als7            ;Same Allocation Vector Size
css10   equ     css7            ;Same Checksum Vector Size
xlt10   equ     xlt7            ;Same Translate Table
;               DISKDEF 11,7
dpb11   equ     dpb7            ;Equivalent Parameters
als11   equ     als7            ;Same Allocation Vectr Size
css11   equ     css7            ;Same Checksum Vector Size
xlt11   equ     xlt7            ;Same Translate Table
;               ENDEF
;
;       Uninitialized Scratch Memory Follows:
;
begdat  equ     offset $        ;Start of Scratch Area
dirbuf  rs      128             ;Directory Buffer
alv0    rs      als0            ;Alloc Vector
csv0    rs      css0            ;Check Vector
alv1    rs      als1            ;Alloc Vector
csv1    rs      css1            ;Check Vector
alv2    rs      als2            ;Alloc Vector
csv2    rs      css2            ;Check Vector
alv3    rs      als3            ;Alloc Vector
csv3    rs      css3            ;Check Vector
```

```
alv4    rs      als4            ;Alloc Vector
csv4    rs      css4            ;Check Vector
alv5    rs      als5            ;Alloc Vector
csv5    rs      css5            ;Check Vector
alv6    rs      als6            ;Alloc Vector
csv6    rs      css6            ;Check Vector
alv7    rs      als7            ;Alloc Vector
csv7    rs      css7            ;Check Vector
alv8    rs      als8            ;Alloc Vector
csv8    rs      css8            ;Check Vector
alv9    rs      als9            ;Alloc Vector
csv9    rs      css9            ;Check Vector
alv10   rs      als10           ;Alloc Vector
csv10   rs      css10           ;Check Vector
alv11   rs      als11           ;Alloc Vector
csv11   rs      css11           ;Check Vector
enddat  equ     offset $        ;End of Scratch Area
datsiz  equ     offset $-begdat ;Size of Scratch Area
        db      0               ;Marks End of Module
```

APPENDIX E

PROGRAM LISTING OF RXFLOP.A86

```
;Prog Name   : RXFLOP.A86 (REMEX FLOPPY DISK
;                                 ACCESS CODE)
;Date        : 5 April 1983
;Modified by: Mark L. Perry
;For         : Thesis (AEGIS Modeling Group)
;Advisor     : Professor Cotton
;Purpose     : This code is an include file w/in CPMBIOS.A86.
;              It contains the code necessary to access the
;              Remex floppy disk drives and generate a
;              loader for them.
;              This configuration is set for CP/M logical
;              drives 1 (A: & B:) and 2 (C:). To alter
;              change code in READ and WRITE routines.


;*****************************************************************
;                         Equates
;*****************************************************************

;--- Disk Controller command bytes and masks (REMEX)     ---

    dk_rdy_mask      equu 08H
    dk_rd_cmd1       equ 1011H      ;read command
    dk_rd_cmd2       equ 1012H
    dk_wr_cmd1       equ 1021H      ;write command
    dk_wr_cmd2       equ 1022H
    tries            equ 10
    drive2           equ 2          ;CPM logical dsk # for
                                    ;drive 2


;--------- REMEX Interface  Controller Ports -------------

    cmd_reg          equ 70H        ;ctrler's base in CP/M-86
    status_reg       equ 71H
    p_addr_lo        equ 72H
    p_addr_hi        equ 73H

;*****************************************************************
;                   CPM DEVICE SPECIFIC CODE
;     entered from the main CPMBIOS's jump vectors
;*****************************************************************
```

```
        cseg $
;-----------------------------------------------------------------
rxflop_init:
        ret                     ;no special action required

;-----------------------------------------------------------------
rxflop_home:
        ret                     ;no special action required


;-----------------------------------------------------------------
rxflop_seldsk:
        ret                     ;no special action required


;-----------------------------------------------------------------
rxflop_settrk:
        ret                     ;no special action required


;-----------------------------------------------------------------
rxflop_setsec:
        ret                     ;no special action required


;-----------------------------------------------------------------
rxflop_read:
        mov     rwdir,2
        IF      loader_bios
;-----------------------------------------------------------------
```

```
;
        mov     bx,dk_rd_cmd1       ;force a read on drive 1
;
;------------------------------------------------------------
        ENDIF                       ;loader_bios

        IF      not loader_bios
;------------------------------------------------------------
;
        call    request             ;request ticket number
        cmp     unit,drive2         ;CP/M logical disk No. for
        jz      rd1                 ;Remex floppy drive 2 (C:)
        mov     bx,dk_rd_cmd1       ;set up to rd drive 1 (B:)
        jmps    rd2
    rd1:
        mov     bx,dk_rd_cmd2       ;set up to rd drive 2
    rd2:
;
;------------------------------------------------------------
        ENDIF                       ;not loader_bios
        call    build_packet
        call    send_packet         ;perform the read
        call    xfr_buffer          ;xfr CPM buffer into memory

        IF      not loader_bios
;------------------------------------------------------------
;
        call    release             ;free resource
;
;------------------------------------------------------------
        ENDIF                       ;not loader_bios

        mov     al,result           ;return success/failure code
        ret


;------------------------------------------------------------

rxflop_write:

        IF      not loader_bios
;------------------------------------------------------------
;
        mov     rwdir,1
        call    request             ;request ticket number
        cmp     unit,drive2         ;CP/M logical disk No. for
        jz      wrt1                ;Remex floppy drive 2 (C:)
        mov     bx,dk_wr_cmd1       ;setup write to drive 1 (B:)
        jmps    wrt2
    wrt1:
        mov     bx,dk_wr_cmd2       ;set up to write drive 2
```

```
wrt2:
        call    build_packet
        call    xfr_buffer
        call    send_packet
        call    release         ;free resource
        mov     al,result       ;return success/failure code
;
;-------------------------------------------------------------
        ENDIF                   ;not loader_bios

        ret



;*************************************************************
;                REMEX FLOPPY DISK SUBROUTINES
;*************************************************************

build_packet:
        push    es              ;save es register
        mov     ax,cmemseg      ;set up es to address common
        mov     es,ax           ;memory
        mov     p_modifiers,bx  ;enter read code in packet
        mov     p_status,0      ;clear packet status word
        mov     ax,0000H        ;clear register
        mov     al,track        ;get track #
        mov     p_track_no,ax   ;enter track # in packet
        mov     ax,0000H        ;set head no. to 0
        add     al,sector       ;set sector no.
        mov     p_head_sect,ax  ;put head & sec # in packet
        mov     p_mem_addr,0100h;address of CPM buffer
        mov     p_msb,000eh     ;CPM buffer msb
        mov     p_word_count,64 ;# of 16 bit words
        pop     es
        ret

;------------------------------------------------------------

send_packet:

        push    es
        mov     ax,cmemseg
        mov     es,ax
        mov     dk_cnt,tries   ;load count for retries
   send1:
        in      al,status_reg
        and     al, dk_rdy_mask ;check interface ready
        cmp     al,08H          ;is it ready?
        jne     send1           ;if not ready repeat
        mov     al,1cH
```

```
        out     cmd_reg,al          ;load extended address
        mov     ax,0004h            ;packet offset
        out     p_addr_lo,al        ;transfer low byte out
        mov     al,ah
        out     p_addr_hi,al        ;transfer hi byte out
check_result:
        mov     ax,p_status         ;load status word
        cmp     ax,0001H            ;check for success
        je      success_read
        cmp     ax,0000H            ;check for failure
        jne     retry
        jmps    check_result
    retry:
        mov     dk_err_code,al      ;save error code
        mov p_status,0              ;clear status word
        dec     dk_cnt              ;reduce retry count
        jnz     send1               ;if <> 0 try again
        mov     result,0FFH         ;return failure code
        jmps    dk_execute_ret
    success_read:
        mov     result,00H                  ;return success code
    dk_execute_ret:
        pop     es
        ret

;-----------------------------------------------------------

xfr_buffer:

        push es ! push ds
        mov  es,dma_seg
        mov  di,dma_adr
        mov  ax,cmemseg
        mov  ds,ax
        mov  si,0100h
        mov  cx,64
        cmp  rwdir,0
        jz   xfr
        xchg si,di                  ;set up for write operation
        mov  ax,ds
        mov  es,ax
        mov  ds,dma_seg
xfr:
        cld
        rep  movs ax,ax             ;move as 16-bit words
        pop  ds ! pop es
        ret

;**********************************************************
;                   Data Segment Area
;**********************************************************
```

```
;----------------- Remex Interface Packet----------------

                 eseg
                 org 0024h  ;offset of packet

p_modifiers      rw  1      ;function & logical unit
p_status         rw  1      ;returned status
p_track_no       rw  1      ;selected track number
p_head_sect      rw  1      ;selected head/sector number
p_mem_addr       rw  1      ;buffer address
p_msb            rw  1      ;extended bits of buffer address
p_word_count     rw  1      ;size of data block

;-------------------Misc Variables----------------------

                 cseg $

dk_err_code      db  00H    ;returned Remex error code
dk_cnt           db  00H


result           rb  1
rwdir            rb  1      ;0 = read ; 1 = write
```

APPENDIX I

PROGRAM LISTING OF LDRMAST.CFG

```
;Prog Name     :LDRMAST.CFG
;Date          :5 April 1983
;Written by    :Mark L. Perry
;For           :Thesis (AEGIS Modeling Group)
;Advisor       :Professor Cotton
;Purpose       :This code is an include file within
;               LDCPM.A86. It contains the device
;               tables for access to initialization,
;               read, and write routines and was
;               developed to accompany the boot rom
;
;------------------------------------------------------------
;                    DEFINE nunits
nunits  db 1    ;only a single drive for the loader
;
;------------------------------------------------------------
;                    INITIALIZATION TABLE
intbl   dw offset rxflop_init   ;initialize Remex
        dw 0
;
;------------------------------------------------------------
;                    ADDITIONAL OFFSETS
;
rdtbl   dw offset rxflop_read
wrtbl   dw offset rxflop_write
hmtbl   dw offset rxflop_home
dsktbl  dw offset rxflop_seldsk
trktbl  dw offset rxflop_settrk
sectbl  dw offset rxflop_setsec
;
;------------------------------------------------------------
```

APPENDIX J

PROGRAM LISTING OF LDRMAST.DEF

This is the disk definition statement used for the loader routine. The system is configured in the loading phase as a single disk system to minimize the space requirements.

```
disks 1
diskdef 0,1,26,6,1024,243,64,64,2
endef
```

APPENDIX K

PROGRAM LISTING OF LDRMAST.LIB

```
;                       DISKS 1
dpbase    equ           $                   ;Base of Disk Parameter Blocks
dpe0      dw            xlt0,0000h          ;Translate Table
          dw            0000h,0000h         ;Scratch Area
          dw            dirbuf,dpb0         ;Dir Buff, Parm Block
          dw            csv0,alv0           ;Check, Alloc Vectors
;                       DISKDEF 0,1,26,6,1024,243,64,64,2
dpb0      equ           offset $            ;Disk Parameter Block
          dw            26                  ;Sectors Per Track
          db            3                   ;Block Shift
          db            7                   ;Block Mask
          db            0                   ;Extnt Mask
          dw            242                 ;Disk Size - 1
          dw            63                  ;Directory Max
          db            192                 ;Alloc0
          db            0                   ;Alloc1
          dw            16                  ;Check Size
          dw            2                   ;Offset
xlt0      equ           offset $            ;Translate Table
          db            1,7,13,19
          db            25,5,11,17
          db            23,3,9,15
          db            21,2,8,14
          db            20,26,6,12
          db            18,24,4,10
          db            16,22
als0      equ           31                  ;Allocation Vector Size
css0      equ           16                  ;Check Vector Size
;                       ENDEF
;
;         Uninitialized Scratch Memory Follows:
;
begdat    equ           offset $            ;Start of Scratch Area
dirbuf    rs            128                 ;Directory Buffer
alv0      rs            als0                ;Alloc Vector
csv0      rs            css0                ;Check Vector
enddat    equ           offset $            ;End of Scratch Area
datsiz    equ           offset $-begdat     ;Size of Scratch Area
          db            0                   ;Marks End of Module
```

APPENDIX L

PROGRAM LISTING OF RMXROM.A86

```
;*************************************************************
;*                                                          *
;* This Customized ROM loader for CP/M-86 has               *
;* the following hardware configuration:                    *
;*      Processor:  iSBC 86/12a                             *
;*      Disk Controller:  Remex Data Warehouse              *
;*                        or the iSBC 201,202               *
;*      Memory model:  8080                                 *
;*                                                          *
;*      Programmer:  M.L. Perry                             *
;*                                                          *
;*************************************************************

;*********************************************
;* This is the BOOT ROM which is resident   *
;* in the 957 monitor. To execute the boot  *
;* the monitor must be brought on-line and  *
;* then control passed by "gffd4:Ø" or by   *
;* "gffd4:ØØØ4". The first monitor command  *
;* will boot to an iSBC 202 disk and the    *
;* second command will boot to the Remex.   *
;* First, the ROM moves a copy of its data  *
;* to RAM at location ØØØØØH, then it       *
;* initializes the segment registers and the*
;* stack pointer. The i8259 peripheral int- *
;* rupt controller is setup for interrupts  *
;* at 1ØH to 17H (vectors at ØØØ4ØH-ØØØ5FH) *
;* and edge-triggered auto-EOI (end of in-  *
;* terrupt) mode with all interrupt levels  *
;* masked-off.  Next, the appropriate device*
;* controller is initialized, and track Ø   *
;* sector 1 is read to determine the target *
;* paragraph address for LOADER.  Finally,  *
;* the LOADER on track Ø sectors 2-26 and   *
;* track 1 sectors 1-26 is read into the    *
;* target address.  Control then transfers  *
;* to the LOADER program for execution. ROM *
;* Ø contains the even memory locations, and*
;* ROM 1 contains the odd addresses.  BOOT  *
;* ROM uses RAM between ØØØØØH and ØØØFFH   *
;* (absolute) for a scratch area.           *
;*********************************************
```

```
;*********************   EQUATES   ***************************
;*                                                           *

;------------------- Miscellaneous equates --------------------
;|                                                           !
   cr                equ 0dH    ;Ascii carriage return
   disk_type         equ 01H    ;type for iSBC 202 disk
   lf                equ 0aH    ;Ascii line feed
   remex_type        equ 02H    ;type for REMEX floppy
   romseg            equ 0ffd4h ;base address
   sector_size       equ 128    ;CP/M sector size
   start_trk1        equ 0c8h
;|                                                           !
;-------------------------------------------------------------

;-------------- I8251 USART console ports ----------------
;|                                                           !
   CONP_data         equ 0d8H   ;I8251 data port
   CONP_status       equ 0daH   ;I8251 status port
;|                                                           !
;-------------------------------------------------------------

;--- Disk Controller command bytes and masks (iSBC 202) ---
;|                                                           !
   DK_chkint_mask    equ 004H   ;mask to check for DK interupt
   DK_home_cmd       equ 003H   ;move to home position command
   DK_read_cmd       equ 004H   ;read command
;|                                                           !
;-------------------------------------------------------------

;-------- INTEL iSBC 202 Disk Controller Ports ----------
;|                                                           !
   DKP_base          equ 078H          ;ctrler's base in CP/M-86
   DKP_result_type   equ DKP_base+1 ;operation result type
   DKP_result_byte   equ DKP_base+3 ;operation result byte
   DKP_reset         equ DKP_base+7 ;disk reset
   DKP_status        equ DKP_base   ;disk status
   DKP_iopb_low      equ DKP_base+1 ;low addr byte of iopb
   DKP_iopb_high     equ DKP_base+2 ;high addr byte of iopb
;|                                                           !
;-------------------------------------------------------------

;---------REMEX floppy disk drive equates------------------
;|                                                           !
```

```
        dk_rd_cmd1     equ 1011h          ;read for drive 1
        dk_rdy_mask       equ 08h           ;ready mask for control
        cntrl_err_mask   equ 04h           ;controller error
        d_err_mask        equ 08h           ;disk error
        crc_err_mask      equ 10h           ;crc error
        cmemseg           equ 0e000h        ;common memory
        tries             equ 10            ;number of retries
;|                                                              |
;--------------- REMEX Controller Ports ------------------
;|                                                              |
cmd_reg                 equ     70h      ;controller base
status_reg              equ     71h      ;status register
p_addr_lo               equ     72h      ;lower address
p_addr_hi               equ     73h      ;upper address
;|                                                              |
;---------------------------------------------------------


;----- INTEL i8259 Programmable Interrupt Controller ------
;|                                                              |
PIC_59p1                equ     0C0h     ;8259a port 0
PIC_59p2                equ     0C2h     ;8259a port 1
;|                                                              |
;---------------------------------------------------------


;
;************** ENTRY POINT AND MAIN CODE ***************
;
        cseg   romseg
;
;Enter here with gffd4:0 command for iSBC 202 boot
;
        mov DL,disk_type              ;set boot type to disk
        jmps Start_boot               ;go start code
;
;Enter here with gffd4:0004 command for REMEX boot
;
t_1:
        mov DL,remex_type             ;set boot type to remex
Start_boot:
;
;move our data area into RAM at 0000:0200
;
        mov AX,CS                     ;point DS to CS for source
        mov DS,AX
        mov SI,databegin              ;start of data
        mov DI,offset ram_start       ;offset of destination
        mov AX,0                      ;set dest segment (ES)
        mov ES,AX                     ;to 0000
        mov CX,data_length            ;how much to move (bytes)
```

```
        rep movs AL,AL              ;move from eprom,
                                    ;byte at a time
;
;set segment registers and initialize the stack
;
        mov AX,0                    ;set DS segment to 0000.
                                    ;now in RAM
        mov DS,AX                   ;data segment now in RAM
        mov SS,AX
        mov SP,stack_offset         ;init stack segment/pointer
        cld                         ;clear the direction flag
;
;Setup the 8259 Programmable Interrupt Controller
;
        mov AL,013H
        out PIC_59p1,AL    ;8259a ICW 1  8086 mode
        mov AL,010H
        out PIC_59p2,AL    ;8259a ICW 2  vector @ 40-5F
        mov AL,01fH
        out PIC_59p2,AL    ;8259a ICW 4  auto EOI master
        mov AL,0ffH
        out PIC_59p2,AL    ;8259a OCW 1  mask all levels off
;
;************ BRANCH TO SELECTED DEVICE FOR BOOT **********
;
;   determine if booting to iSBC 202 or to REMEX
;
        cmp DL,disk_type            ;is this a i202?
        jne boot_remex              ;if not, boot to REMEX
;
;***************** iSBC 202 BOOT CODE *********************
;
Boot_i202:                          ;return on fatal errors
;
;Rest and initialize the iMDS 800 Diskette Interface
;
        in  AL,DKP_result_type      ;clear the controller
        in  AL,DKP_result_byte
        out DKP_reset,AL            ;AL is dummy
                                    ;for this command
;
; home the iSBC 202
;
        mov DK_io_com,DK_home_cmd ;load io command
        call DK_Execute_Cmd         ;home the disk
        mov DK_io_com,DK_read_cmd ;all io now reads only
;
; get track 0, sector 1, the GENCMD header record
;
        mov BX,offset genheader ;offset for 1st sector DMA
        mov DK_dma_addr,BX        ;store dma address in iopb
```

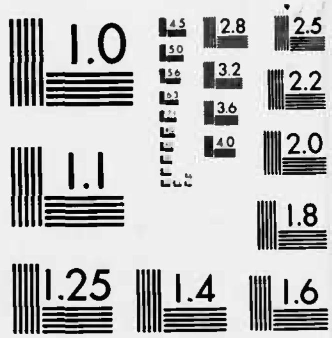
```
        mov DK_secs_tran,1       ;transfer 1 sector
        mov DK_sector,1          ;start at sector #1
        call DK_Execute_Cmd      ;read track 0, sector 1
;
; get trk 0, sect 2-26, and place in RAM
;
        mov ES,abs_location      ;segment loc for LOADER
        mov AX,ES                ;to AX to manipulate
        mov CL,04                ;must xlat to 16-bit addr
        sal AX,CL                ;shift segment
        mov DK_dma_addr,AX       ;store dma address in iopb
        mov DK_secs_tran,25      ;transfer 25 sectors
        mov DK_sector,2          ;start at sector #2
        call DK_Execute_Cmd      ;read trk 0, sects 2-26
;
; get trk 1, sect 1-26, put at next place in RAM
;
        mov AX,ES                ;compute offset for track 1
        add AX,start_trk1        ;add in what already read
        mov CL,04                ;must xlat to 16-bit addr
        sal AX,CL                ;shift segment
        mov DK_dma_addr,AX       ;store dma address in iopb
        mov DK_secs_tran,26      ;transfer 26 sectors
        mov DK_sector,1          ;start at sector #1
        mov DK_track,1           ;start at track #1
        call DK_Execute_Cmd      ;read trk 1, sects 1-26
        jmp Jump_To_Loader       ;go pass control to loader
;
;******************** REMEX BOOT CODE ********************
;
boot_remex:
        mov AX,cmemseg           ;make common memory
        mov ES,AX                ;addressable
;
; get track 0, sector 1, the GENCMD header
;
boot_again:                      ;return here on errors
        mov BX,dk_rd_cmd1        ;set up to read drive 1
        mov ah,00h               ;track 0
        mov al,01h               ;sector 1
        push ax                  ;save it
        call build_packet        ;do it
        call send_packet
        mov di,offset genheader ;set up destination
        mov ax,ds                ;set up source
        call xfer_buffer
;
; get load location from GENCMD header
;
;
        mov BX,abs_location      ;abs is location in RAM
```

```
        mov CL,04                    ;convert to 16 bit
        sal BX,CL                    ;BX now has load address
        mov DI,BX                    ;now in di
get:
        pop ax                       ;get next sector
        inc al                       ;and check for last
        cmp al,26                    ;on this track
        ja fin_1
get_1:
        push ax                      ;save ax
        mov BX,dk_rd_cmd1            ;prepare for read
        call build_packet            ;read next sector
        call send_packet
        mov ax,0000h                 ;absolute load
        call xfer_buffer
        jmp get
fin_1:
        inc ah                       ;get next track?
        cmp ah,1
        ja Jump_To_Loader            ;jump to loader
        mov al,1                     ;reset sector number
        jmp get_1
;*************** PASS CONTROL TO LOADER ******************
;
    Jump_To_Loader:
        mov ES,abs_location          ;segment addr of LOADER
        mov leap_segment,ES          ;load
      ;setup far jump vector
        mov leap_offset,0            ;offset of LOADER
        jmpf dword ptr leap_offset
;
;****************** END OF MAIN CODE ********************



;*********** BEGINNING OF SUBROUTINES ********************

;**********************************************************
;*                    CONIN  subroutine                  *
;**********************************************************
                    ;called from: Dk_Execute_Cmd.
Conin:              ;** returns console keyboard character
                    ;** parm in - none
                    ;** parm out - returns character in AL
        in AL,CONP_status ;get status
        and AL,2          ;see if ready-bit 1-is set
        jz Conin          ;if not, it is zero and not ready
        in AL,CONP_data ;ready, so read character
        and AL,07fH       ;remove parity bit
```

```
        ret
;
;
;*****************************************************************
;*                  CONOUT  subroutine                            *
;*****************************************************************
                    ;called from: Print_Msg.
Conout:             ;** write character to console keyboard.
                    ;** parm in - character to be output in CL
                    ;** parm out - none
        in AL,CONP_status ;get console status
        and AL,1          ;see if ready-bit 0-is set
        jz CONOUT         ;if zero, not ready-keep checking
        mov AL,CL         ;load input parm to AL for out
        out CONP_data,AL ;output character to console
        ret
;
;
;
;
;*****************************************************************
;*               DK_EXECUTE_CMD  subroutine                       *
;*****************************************************************

                    ;called from: in-line from Boot_1202.
Dk_Execute_Cmd:     ;** Executes a disk read/write command
                    ;** parm in - DMA addr in BX.
                    ;** parm out - none
    ;send iopb to disk controller via two ports (2 bytes)
   Send_iopb:
        in AL,DKP_result_type ;clear the controller
        in AL,DKP_result_byte ;clear the controller
        mov AX,offset DK_iopb ;get address of iopb
        out DKP_iopb_low,AL ;output low byte of iopb addr
        mov AL,AH         ;load high byte to AL for output
        out DKP_iopb_high,AL ;out high byte of iopb addr
    ;check for interrupt from disk controller
   Disk_int:
        in AL,DKP_status ;get disk status
        and AL,DK_chkint_mask ;interrupt set?
        jz Disk_int       ;if not, keep checking
    ;see if interrupt signifies I/O completion
        in AL,DKP_result_type ;get reason for interrupt
        cmp AL,00H        ;was I/O co plete ?
        jz Check_result ;if so, go check the result byte
        jmps Send_iopb  ;if not, go try again
    ;check result byte for errors
   Check_result:
        in AL,DKP_result_byte ;get result byte
        and AL,080H       ;is I/O complete?
        jnz Fatal_err   ;if not, fatal error
```

AD-A132 165 LOGIC DESIGN OF A SHARED DISK SYSTEM IN A MULTI-MICRO COMPUTER ENVIRONMENT(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA M L PERRY JUN 83 3/3

UNCLASSIFIED F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
        and AL,0feH        ;check for error in any bit
        jz  DK_execute_ret ;no errors, go return
   Fatal_err:
        mov CL,0           ;clear CL for counter
   Ftest:
        rcr AL,1           ;check each bit of result
        inc CL             ;count each bit
        test AL,01         ;test each bit
        jz Ftest           ;zero, go check next
        mov AL,CL          ;not zero, error, inc count
        mov AH,0           ;clear high
        add AX,AX          ;double for idx to word table
        mov BX,AX          ;load BX as index
        mov BX,errtbl[BX] ;get addr of error msg
     ;print appropriate error message
        call Print_Msg  ;write msg to console
        call Conin         ;wait for key strike
        jmp Boot_i202   ;then start all over
   Dk_execute_ret:
        ret
;
;
;**************************************************************
;*          REMEX   send_packet subroutine                   *
;**************************************************************
;
send_packet:
        mov ES:dk_cnt,tries        ;load count for retries
send_1:
        in AL,status_reg
        and AL,dk_rdy_mask         ;check interface ready
        cmp AL,08h                 ;ready?
        jne send_1                 ;if not, repeat
        mov AL,1ch                 ;load extended address
        out cmd_reg,AL
        mov AX,0004h               ;packet offset
        out p_addr_lo,AL           ;transfer lo byte
        mov al,ah
        out p_addr_hi,al           ;transfer hi byte
ck_result:
        mov AX,p_status            ;load status
        cmp AX,0001h               ;check for success
        je success_read
        cmp AX,0000h               ;check for failure
        jne retry
        jmps ck_result  ;not finished
retry:
        mov p_status,0000h         ;clear status code
        dec ES:dk_cnt              ;re uce retry count
        jnz send_1
        mov BX,offset errtbl
```

```
        add BX,14                   ;adjust for table entry
        mov BX,[BX]
        call Print_Msg
        call Conin                  ;wait on user key strike
        jmp boot_again              ;start over
success_read:
;
        ret
;
;*                                                              *
;****************************************************************
;
;
;****************************************************************
;*              REMEX build_packet subroutine                   *
;****************************************************************
;
build_packet:
        mov ES:p_modifiers,bx   ;set read code
        mov ES:p_status,0       ;clear status word
        mov dx,0000h            ;clear dx
        mov dl,ah               ;set track number
        mov ES:p_track_no,dx    ;enter in packet
        mov dx,0000h            ;clear dx
        mov dl,al               ;set sector and head
        mov ES:p_head_sect,dx   ;enter in packet
        mov ES:p_mem_addr,0100h ;address of buffer
        mov ES:p_msb,000eh      ;bu fer msb
        mov ES:p_word_count,64  ;number of 16 bit wds
        ret
;*                                                              *
;****************************************************************
;****************************************************************
;*          REMEX xfer_buffer subroutine                        *
;****************************************************************
;
xfer_buffer:
        push es!push ds         ;save segment registers
        mov es,ax               ;set up for transfer
        mov ax,cmemseg
        mov ds,ax
        mov si,0100h            ;location of buffer
        mov cx,64               ;word count
        cld                     ;clear direction flag
        rep movs AX,AX
        pop ds!pop es
        ret
;
;****************************************************************
;
;****************************************************************
```

```
;*               PRINT_MSG  subroutine                              *
;*********************************************************************
                  ;called from: Dk_Execute_Cmd.
Print_Msg:        ;** Prints a message to the console.
                  ;** parm in - address of message in BX.
                  ;** parm out - none
       mov CL,[BX]      ;get next char from message
       test CL,CL       ;is it zero - end of message ?
       jz Pmsg_ret      ;if zero return
       push BX          ;save address of message
       call Conout      ;print it
       pop BX           ;restore address of message
       inc BX           ;next character in message
       jmps Print_Msg   ;next character and loop
Pmsg_ret:
       ret
;
;
;*************** END OF SUBROUTINES *************************


;Image of data to be moved to RAM
;
databegin equ offset $
;
;A template iSBC 202 iopb (channel command - 7 bytes)
                  db  80H    ;iopb channel word
                  db  0      ;io command
                  db  0      ;number of sectors to xfer
                  db  0      ;track to read
                  db  0      ;sector to read
                  dw  0000H  ;dma addr for iSBC 20_
;End of iopb
;
cerrtbl dw      offset er0
        dw      offset er1
        dw      offset er2
        dw      offset er3
        dw      offset er4
        dw      offset er5
        dw      offset er6
        dw      offset er7
;
Cer0    db      cr,lf,'Null Error?',0
Cer1    db      cr,lf,'CRC Error',0
Cer2    db      cr,lf,'Seek Error',0
Cer3    db      cr,lf,'Address Error',0
Cer4    db      cr,lf,'Data Overrun-Underrun',0
Cer5    db      cr,lf,'Write Protect',0
Cer6    db      cr,lf,'Write Error',0
```

```
Cer7    db      cr,lf,'Drive Not Ready',0
;
dataend equ offset $
;
data_length     equ dataend-databegin
;
;       reserve space in RAM for data area
;       (no hex records generated here)
;
        DSEG    0
        org     0200H
;
ram_start       equ     $
;
;This is the iSBC 202 iopb (channel command - 7 bytes)
DK_iopb         rb  1       ;iopb channel word
DK_io_com       rb  1       ;io command
DK_secs_tran    rb  1       ;number of sectors to xfer
DK_track        rb  1       ;track to read
DK_sector       rb  1       ;sector to read
DK_dma_addr     rw  1       ;dma addr for iSBC 202
;End of iopb
;
errtbl          rw      8
er0             rb      length cer0     ;16
er1             rb      length cer1
er2             rb      length cer2
er3             rb      length cer3
er4             rb      length cer4     ;14
er5             rb      length cer5     ;11
er6             rb      length cer6     ;15
er7             rb      length cer7     ;17
;
leap_offset     rw      1
leap_segment    rw      1
;
                rw      32      ;local stack
stack_offset    equ     offset $;stack from here down
;
;128 byte sector will be read in here-GENCMD header
genheader       equ offset $
                rb      1
                rw      1
abs_location    rw      1  ;absolute load location
                rw      1
                rw      1


;***********************************************************
;*           REMEX packet and data locations              *
;***********************************************************
```

```
;
        ESEG
        org 0004h                       ;offset of REMEX packet
p_modifiers     rw      1               ;function and logic unit
p_status        rw      1               ;returned status
p_track_no      rw      1               ;selected track number
p_head_sect     rw      1               ;selected head/sector number
p_mem_addr      rw      1               ;buffer address
p_msb           rw      1               ;extended bits of buffer addr
p_word_count    rw      1               ;size of data block
;
; org higher than buffer to be sure
;
        org 0500h
dk_cnt          rb      1               ;number retries
;
;*****************************************************************
;*              End of CP/M-86 Customized ROM                   *
;*****************************************************************
                END
;
```

LIST OF REFERENCES

1. Wasson, W.J., Detailed Design of the Kernel of a Real-Time Multiprocessor Operating System, M.S. Thesis, Naval Postgraduate School, June 1980.

2. Klinefelter, S.G., Implementation of a Real-Time, Distributed Operating System for a Multiple Computer System, M.S. Thesis, Naval Postgraduate School, June 1982.

3. Candalor, M.B., Alteration of the CP/M-86 Operating System, M.S. Thesis, Naval Postgraduate School, June 1981.

4. Hicklin, M.S. and Neufeld, J.A., Adaptation of Magnetic Bubble Memory in a Standard Microcomputer Environment, M.S. Thesis, Naval Postgraduate School, December 1981.

5. Almquist, T.V. and Stevens, D.S., Alteration and Implementation of the CP/M-86 Operating System for a Multi-user Environment, M.S. Thesis, Naval Postgraduate School, December 1982.

6. INTEL Corporation, iAPX 86,88 User's Manual, 1981.

7. INTEL Corporation, iSBC 86/12A Single Board Computer Hardware Reference Manual, 1979.

8. EX-CELL-O Corporation, REMEX Technical Manual for Data Warehouse Models RDW 3100, RDW 3200, 1979.

9. INTEL Corporation, iCS-80 Industrial Chassis Hardware Reference Manual, 1979.

10. Digital Research, CP/M-86 Operating System System Guide, 1981.

11. Digital Research, CP/M-86 Operating System User's Guide, 1981.

12. Digital Research, CP/M-86 Operating System Programmer's Guide, 1981.

13. MICROPOLIS Corporation, 1220 Rigid Disk Drive Subsystems, 1981.

14. John, J.W., Utilizing the Micropolis Disk Drive as a System Resource, M.S. Thesis, Naval Postgraduate School, October 1982.

15. INTEL Corporation, Component Data Catalog, 1982.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 62 Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	1
4.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
5.	Professor Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93940	3
6.	Associate Professor M. L. Cotton, Code 62Cc Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	3
7.	LCDR Ron Modes, USN, Code 52Mf Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
8.	CAPT James L. Johnston 378D Bergin Drive Monterey, California 93940	1
9.	Defense Logistic Studies Information Exchange U. S. Army Logistics Management Center Fort Lee, Virginia 23801	1
10.	Daniel Green (Code N20E) Naval Surface Warfare Center Dahlgren, Virginia 22449	1

11. CDR J. Donegan, USN 1
PMS 400B5
Naval Sea Systems Command
Washington, DC 22362

12. RCA AEGIS Data Repository 1
RCA Corporation
Government Systems Division
Mail Stop 127-327
Moorestown, New Jersey 08057

13. Library (Code E33-05) 1
Naval Surface Warfare Center
Dahlgren, Virginia 22449

14. Dr. M. J. Gralia 1
Applied Physics Laboratory
John Hopkins Road
Laurel, Maryland 20707

15. Dana Small 1
NOSC, Code 8242
San Diego, California 92152

16. CPT. Todd B. Kersh 1
P.O. Box 175
Fort Monmouth, New Jersey 07703

17. CPT. Mark L. Perry 2
2922 State Route 235
Xenia, Ohio 45385

END

FILMED

9-83

DTIC